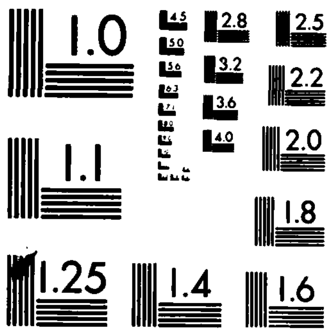
AD-A138 118 CONTINUED DEVELOPMENT OF A UNIVERSAL NETWORK INTERFACE DEVICE USING THE I..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. W F MATHESON 1/2

UNCLASSIFIED 15 DEC 83 AFIT/GE/EE/83D-42 F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

CONTINUED DEVELOPMENT OF A UNIVERSAL NETWORK INTERFACE DEVICE USING THE INTEL 8086 AND 8089 16-BIT MICROPROCESSORS

THESIS

AFIT/GE/EE/83D-42 William F. Matheson
Capt USAF

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY (ATC)

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/EE/83D-42

CONTINUED DEVELOPMENT OF A UNIVERSAL NETWORK INTERFACE DEVICE USING THE INTEL 8086 AND 8089 16-BIT MICROPROCESSORS

THESIS

AFIT/GE/EE/83D-42 William F. Matheson
Capt USAF

AFIT/GE/EE/83D-42

CONTINUED DEVELOPMENT OF A UNIVERSAL NETWORK INTERFACE DEVICE USING THE INTEL 8086 AND 8089 16-BIT MICROPROCESSORS

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

by

William F. Matheson

Capt, USAF

Graduate Electrical Engineering

December 1983

Approved for public release; distribution unlimited.

Preface

This research effort describes the prototype development of an improved Universal Network Interface Device (UNID II). The UNID II's architecture was based on a preliminary design project at the Air Force Institute of Technology. The UNID II contains two modules; a local module and a network module. The operation of both modules is controlled by a single 16-bit 8086 microprocessor. The network module is a remote cluster of two 8089 Input/Output Processors (IOP) which handle the I/O operation for two high speed serial channels. This report documents the detailed design of the network module, plus construction and testing of the local module.

I would like to thank my thesis advisor, Dr. Gary Lamont, for his encouragement and assistance throughout the course of this investigation. I would also like to thank my reader, Major Charles W. Lillie, for his valuable comments and aid during this project. The excellent technical support by the laboratory technicians was greatly appreciated. I also thank my fellow investigators in room 67 who gave me valuable encouragement and assistance when I needed it most. Finally, I wish to express my deepest appreciation to my wife, Nita, and daughters, Janice and Barbra, for their encouragement, assistance, and understanding during my entire graduate program.

Contents

List of Figures

List of Tables

Abstract

This research describes the development of a Universal Network Interface Device (UNID II) which is intended to function as a node in a computer communications network. The UNID II is a 16-bit, 8086 microprocessor based version of the present 8-bit Z80A UNID being developed at the Air Force Institute of Technology (AFIT). The UNID II's architecture was based on a conceptual block diagram design presented in a previous AFIT thesis. It is comprised of two modules: a local module, which interfaces the UNID II to host computers and/or peripheral devices; and a network module, which interfaces the UNID II to a computer communications network. In this report the detailed design of the network module, and the construction and testing of the local module is documented. The network module was designed using a pair of 8089 Input/Output Processors in a remote configuration. The local module consists of an Intel SBC 86/12A single board computer and a wire wrap card with four low speed I/O ports. Testing was done using an Intel ICE-86A/88A In-Circuit Emulator. The tests conducted, verified the proper operation of the local module, including some software to process X.25 formatted frames. The UNID II was not tested in a computer communications network environment.

I Introduction

The purpose of this investigation is to continue the development of an advanced universal network interface device (UNID) using the Intel 8086 and 8089 microprocessors. The UNID ininiall1 was designed and an iSBC 86/12 card (Ref 3,10,21) within a three board multiprocessor configuration. This effort represents another phase of develop nt of the AFIT Digital Engineering Laboratory Network (I NET)(Ref 9, 11, 22, 23,).

The remaining sections of this chapter address background information, the scope of the effort, approach, and an overview of the work covered by this thesis effort.

Background

The 1842 Electronics Engineering Group (EEG) completed a report on 31 Oct 77 to provide an engineering input to the Air Force Communications Command (AFCC) planning process for future base-level telecommunications requirements (Ref 1). The report reviewed the current capability and studied known requirements. It also included a technological forecast as well as predicting the possible effects of new developments on user requirements until 1990. The 1842 EEG report introduced the concept of a multi-ring network to meet the base-level communications needs. The multi-ring network has the advantage of being easily expanded, flexible, and lower in total cost compared to alternative methods. They saw the

key to the implementation of the multi-ring network as development of five interface devices to connect the various equipment to the rings (Figure 1).

The Rome Air Development Center (RADC) was tasked with defining the interface devices. They decided that each device would have many common features and it should be possible to build a programmable device which could be used for all five cases. The development of a programmable network interface became the focus of a series of efforts at AFIT. The initial design of a universal network interface device based on the Zilog Z-80 microprocessor was done in 1978 (Ref 24) and the hardware for two devices was completed in 1981 (Ref 22).

At the same time a separate effort was started to develop a local network for the Air Force Institute of Technology (AFIT) Digital Engineering Laboratory (DEL). The primary goal was to share resources such as printers, memory, software, and mass storage sevices. The first few efforts (Ref 24, and 9) were primarily theoretical and dealt with local networks and switching algorithms in general but not in specific requirements for the DELNET. A later effort (Ref 11) was more specific and tried to determine the requirements of the main users of the DEL. A serious attempt was made to determine what features were nice to have and those that were necessary.

When the first UNID hardware was built it was used to demonstrate that a net was possible (Ref 22) and the

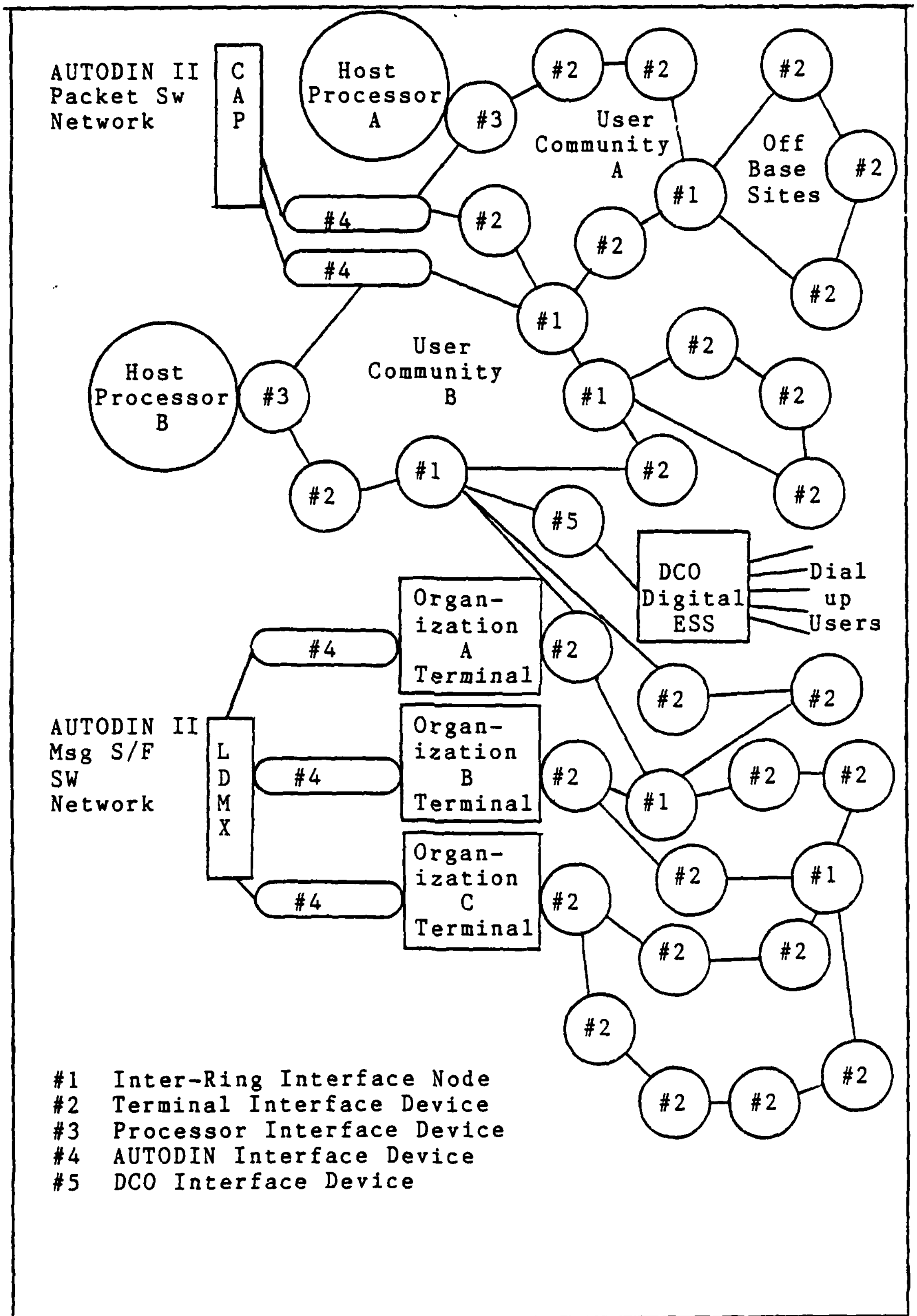


Figure 1. Multi-Ring Base Network (Ref 1)

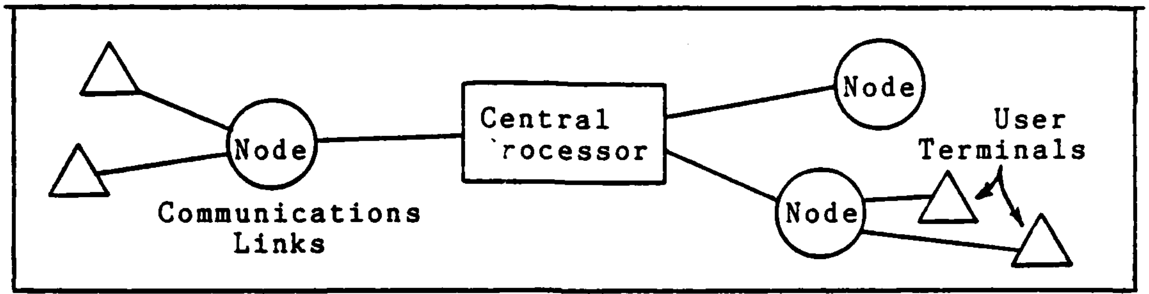


Figure 2 Star Topology

software development that followed was of a very general nature using International Standards Organization (ISO) guidelines and the International Telephone and Telegraph Consultative Committee (CCITT) recommendations. The ISO Open Systems Interconnection (ISO) reference model was chosen and the specific requirements for all levels have been defined (Ref 9,23). The software for the lower three levels will reside in the UNIDs and the other four levels are a function of the host system.

Network Topologies

There are five basic network topologies and many combinations and variations of the basic topologies in general use. The basic types are the star(centralized), loop (ring), tree, mesh (distributed), and bus. Each topology has particular attributes which make it suitable for a particular application or some deficiency to rule it out for other applications. A short discussion of the five basic topologies follows.

In a star network (Figure 2), the overall control is handled by the central processor. The central processor can

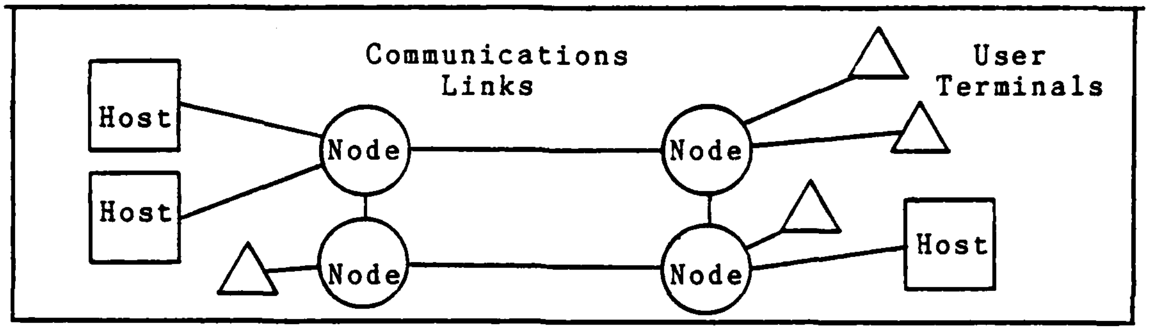


Figure 3. Ring Topology

insure that the networks' resources are fully shared and utilized better than with other topologies. The main disadvantages are that each remote site must have a dedicated commuication link to the central processor and, since all node-to-node communication must pass through the central processor, a central processor failure can disable the entire network.

The ring configuration (Figure 3) is well suited to local networks since it works best when the nodes are relatively close to each other. Each message circulates around the loop and is repeated by each node until the message reaches its destination. The ring minimizes the number of physical communication links required for a net but, since the ring may contain messages from many hosts at any time, this topology usually requires high speed (capacity) transmission links and nodes.

The ring concept can be extended so that many rings are interconnected to form a multi-ring configuration (Figure 1). Each ring can be a relatively self-sufficient building

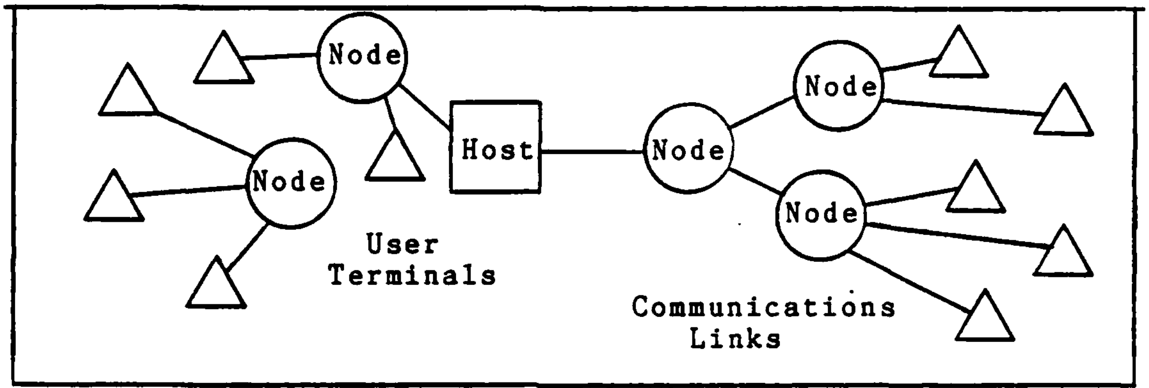


Figure 4. Tree Topology

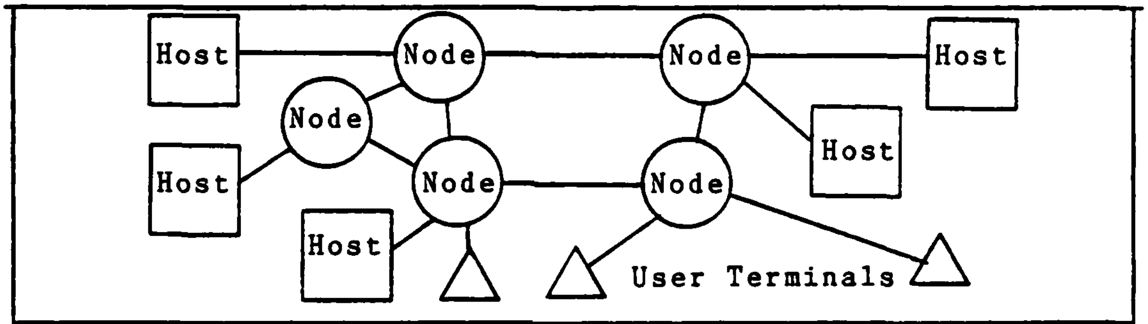


Figure 5. Distributed Topology

block and the network can be expanded as needed by adding rings. Multi-ring topologies have two main advantages; the network cost is distributed among the various users and they do not require sophisticated common control algorithms.

The tree network (Figure 4) is similiar to the star in that it can be controlled by a single computer. The communication links close to the central computer are shared resulting in a reduction in the length of the links when compared with a similiar star network.

The distributed topology (Figure 5) is typical of long haul or interstate networks. Due to their high connectivity, there are usually multiple paths from source to destination. The communication link redundancy increases reliability but

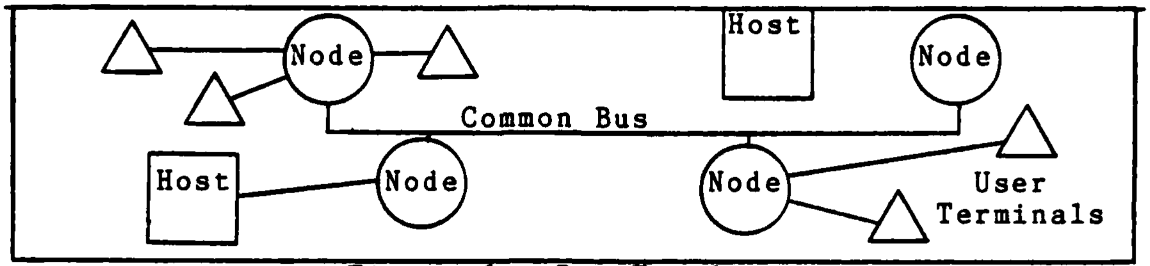


Figure 6. Bus Topology

algorithms for routing and control must be more complex and involved.

Bus architectures (Figure 6) are popular for local networks due to their relative simplicity. Previously discussed topologies must store and retransmit messages at each node along the route from source to destination. In the bus, messages are broadcast to all nodes connected to the common transmission channel (bus). Each message goes to all nodes so care must be taken to insure that only one node is transmitting at any time. There are several methods of collision detection and avoidance in use.

Problem and Scope

Many previous thesis projects have been concerned with the development of a local computer network for the AFIT digital engineering laboratory. An important consideration in developing the LCN is the interfacing of many different types of equipment to the network. The network interface device should be flexible so that additional types of equipment can be added to the net but at the same time it should appear transparant to the users of these systems.

This study was concerned with the continued construction and implementation of a Universal Network Interface Device (UNID) using 8086 series components. This family of components has been designed to allow the tailoring of hardware to meet almost any requirement while minimizing redundent capability (Ref 14:1-3). This UNID is refered to as UNID II since it is entirely different from the previous 8-bit, Z-80A based design, except for the signals on the at the interface to the local systems and to the network.

A block diagram of the UNID II is shown in figure 7, it consists of two sections, the network module to interface with the computer network, and a local module to connect with the various user systems. An off the shelf Intel 86/12A single board computer was the basis of the local module with a serial communications interface board designed and built to provide the input/output from the user systems. Since no boards meeting the requirements for the network card were available off the shelf (Ref 10), a card was designed and constructed in a series of thesis efforts. A block of shared memory is used for all communications between the two modules and the transfer is over the Intel Multibus which is used as the system bus.

Software was developed allowing UNID II to operate in the network. But this is only a small subset of the software required for level three. The intent was to allow testing of the hardware in net. Software for the Z-80 UNID's level three layer is being developed in a concurrent thesis effort

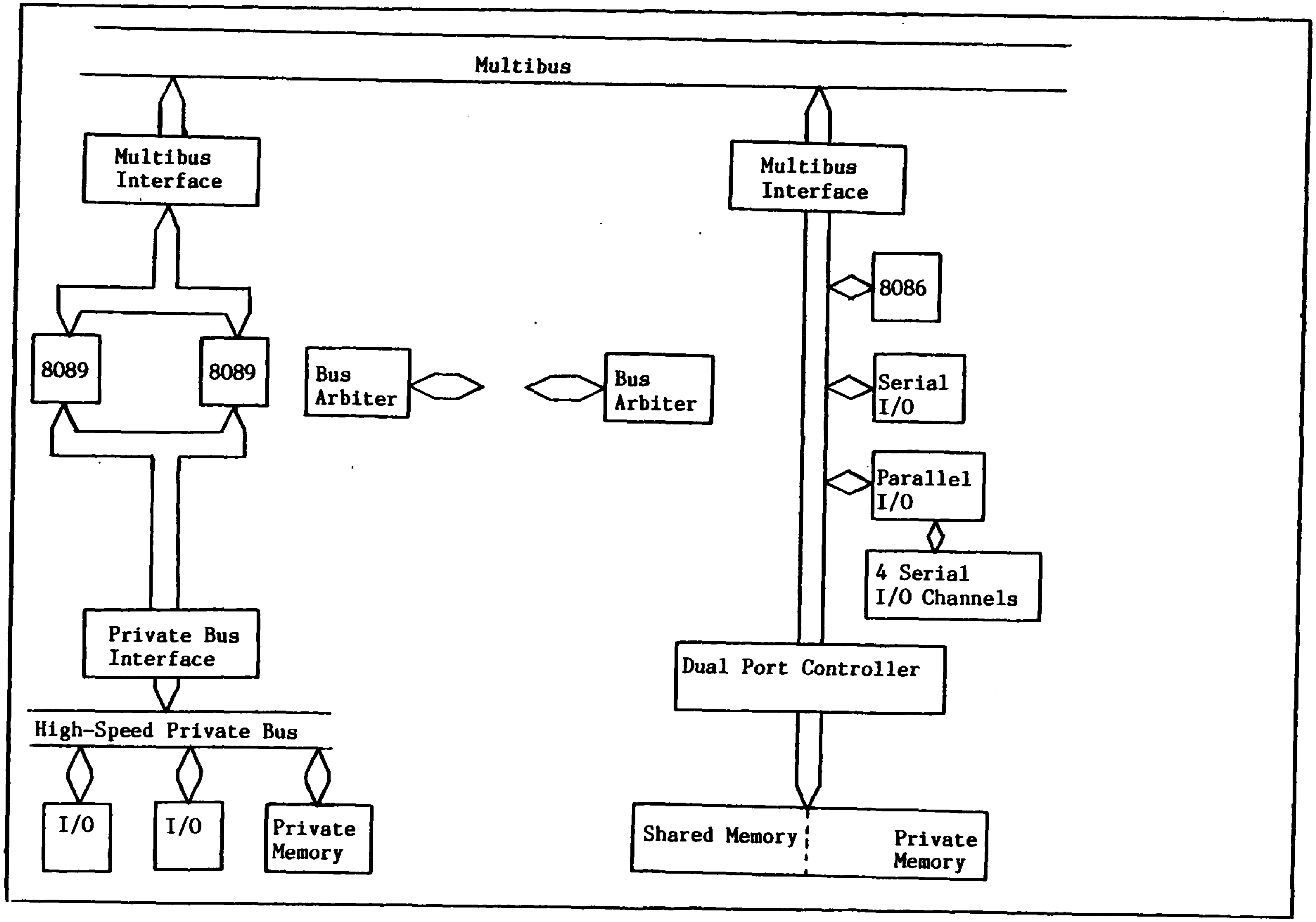


Figure 7. UNID II Block Diagram (Revised)

(Ref 23). These algorithms are being developed in PL/Z (Ref 29) and some of the work may be able to be converted to PL/M-86 (Ref 18) for use on UNID II. The conversion is not within the scope of this effort except as noted in chapter three.

Approach

The first task was to review the work already completed by (Ref 3, 10, and 21) to determine the extent of the task and if there were any hardware limitations or implementation problems. The work done on the original Z-80 based UNID was also studied to insure that UNID II would interface correctly with the present network. Manufacturers' literature for the 8086, 8089, and programable support chips was reviewed to become familiar with the setup requirements and methods.

Prior to adding components to the two prototype boards, local input/output and network, the schematics were updated and all changes were documented. The network board required major hardware changes to provide the handshaking signals needed to interface with the other UNIDs. The local I/O card did not require major changes but it required additional hardware to interface with the host systems.

The actual work on UNID II was done in stages. First, the monitor program for the local processor card was compiled and burned into EPROMs. The associated software to work with this monitor from a host system was developed and tested. Once the monitor program was working additional

circuitry was added to the serial input/output board on the local side, allowing it to interface with RS-232 devices. The wiring was checked for continuity and then power on checks were performed. Software routines were written to test each port for input and output when used with the iSBC 86/12 local processor card. This testing was done on the Intellec system using the ICE-86A in-circuit emulator.

Next, the network board design was modified and additional circuitry added. The network card was changed from a local configuration with an 8086 and an 8089 to a remote cluster of two 8089s. The design was changed to make software development easier and to utilize the special functions provided by the 8089 IOPs.

The last step was the testing of the local subsystem to insure that the software and hardware would operate as a system. Sample packets in datagram format were sent from one local host (terminal) to another. Packets with the destination being another UNID were originated and the memory was checked to see that the proper processing was done. No testing of the network sybsystem was done.

Overview

This report deals with the UNID II hardware and the initial software needed to connect it to the DELnet. Chapter II presents a review of established requirements and design with a discussion of the modifications made during this effort. The hardware additions and software development are

covered in Chapter III. Testing of both hardware and software is documented in chapter IV. The final chapter is a summary of this thesis effort with recommendations for further study.

II. UNID II Requirements

This chapter summarizes the requirements established in previous projects (Ref 4,5,9,10,11,22, and 24). First is a summary of UNID and UNID II requirements. Next, the DELNET requirements are summarized. Then, the UNID/DELNET protocol requirements are discussed with an overview of the standards to be implemented on the UNID II.

UNID Requirements Summary

The original UNID design was based on the following general criteria (Ref 24:13):

- The UNID should function as a store-and-forward concentrator and have message routing capabilities.
- The UNID might require specialized I/O ports for unique communication requirements.
- The UNID should be capable of interfacing to various network operating systems and protocols.

These concepts are still valid and are the primary design goals for the UNID II. The UNID II is currently projected to be used in the DELNET which will be configured as a multi-ring with the host systems forming a star at each node. However, the UNID should be designed to interface with other network configurations which could be implemented later. The UNID hardware should be as flexible as possible so that changes in network protocols can be done in software rather than by changing or redesigning the hardware.

At the lowest, or hardware, level of protocol the inter-

faces have been defined to conform with the EIA RS-232 (Ref 5) and RS-449 (Ref 6) standards. The local interfaces to computers or terminals will use RS-232 and network interfaces will be configured for RS-449. Higher levels of protocol should interface so that changes in the upper levels can be accommodated with changes in UNID software rather than hardware. The various levels should have clearly defined interfaces to make updates and changes easier and faster.

Structured analysis techniques were used to develop the functional requirements of the original UNID (Ref 24). A design using a modular approach was developed. Three separate modules were identified: (1) a local input/output (I/O) module for interfacing the UNID to the user's computers, terminals, or modems; (2) a network I/O module for interfacing the UNID to other UNID's over the network; and (3) a dual processor module for matching the local I/O to the network environment (Ref 24:154-155). The three module types were selected after analysis of the requirements using the Structured Analysis Design Technique (SADT)(Ref 24:11-31).

In 1981, a thesis project was started to design an improved UNID (UNID II). The original functional requirements were used to produce Data Flow Diagrams (DFD) (Ref 10), and a new functional requirements model was developed for UNID II. The result indicated that two distinct groups of requirements were present. One group

dealt with the handling of local messages and the other would handle network messages. While there were many similarities in function, both groups were considered necessary. The input requirements which were used to develop the model are listed in table I. The DFDs served as the basis for the design of UNID II and the diagrams are of sufficient detail to aid in the implementation of UNID II. The original DFDs are reproduced in Appendix A.

DELNET Functional Requirements

A survey of potential DELNET users was taken in 1981 as part of a thesis project (Ref 11:19-23). The responses to the survey were used to formulate a set of functional requirements for the DELNET. A summary of the requirements which were considered to be the most important are listed below:

- -Ability to transfer files accross the network.
- -Ability to share peripherals attached to the hosts on the DELNET.
- -Flexibility with respect to the network topology, protocols, and transmission medium used.
- -Performance monitoring capability.
- -High percentage of availability.
- -User transparency to network configuration and specific operating system of hosts.

Many additional features were identified in the survey but were not considered as important for the initial implementation. Some of the features identified which may be considered in the future include: permit software tool

Table I. UNID II Input Requirements

UNID II Input Requirements:

I. Interface a wide variety of network components and handle various topologies.

   A. Accommodate dissimilar computing equipment
      1) Accomplish code conversion
      2) Perform data-rate speed conversion

   B. Interface peripherals and user terminals to network

   C. Interface host computers to network

   D. Provide a network-to-network interface (gateway)

II. Perform independently of network components

   A. Handle network data transmission and reception
      1) Accommodate network throughput requirements
         a) Provide flow control
      2) Adaptable to different protocols
         a) Handle both synchronous and asynchronous communication
         b) Edit and pack characters into formatted message
         c) Unpack a message
         d) Perform serial to parallel data conversion
         e) Handle error control functions such as Message Acknowledge, No Acknowledge, Repeat, and Timeout
      3) Have error checking and recovery capability

   B. Relieve host computers from network specific functions
      1) Provide a buffer to smooth message traffic
      2) Poll communications lines if they are multidropped
      3) Handle interrupts
      4) Route messages to desired destination
      5) Collect performance, traffic, and error statistics

sharing; ability to perform distributed processing, and work with distributed databases; incorporate fault tolerance; provide a means to connect to other networks, such as ARPANET; connect to the base Cyber 750; and provide security for classified projects.

Using the user generated list of functional requirements, a set of requirements for the DELNET hardware and software was established. A ring topology was selected for the DELNET connections with each node providing a star subnet to the local users. This is that same basic configuration recommended in the communications survey (Ref 1) for base level communications systems except for the star subnet. The communications report seemed to indicate a single user at each network interface while the DELNET requirement is for multiple hosts on each network node. With the ring topology development of routing algorithms should be easier and system expansion simplified, since an elaborate routing scheme is not needed and new nodes can easily be connected to the network.

The system requirements outlined in the thesis project were the basis of a series of additional thesis projects. Development of software procedures and writing of the necessary code was started in 1981 (Ref 9), continued in 1982 (Ref 11), and is part of a concurrent thesis project (Ref 23). At the same time, the UNID hardware design was being refined (Ref 21) and modifications made (Ref 4) both to correct known problems and to try to improve reliability.

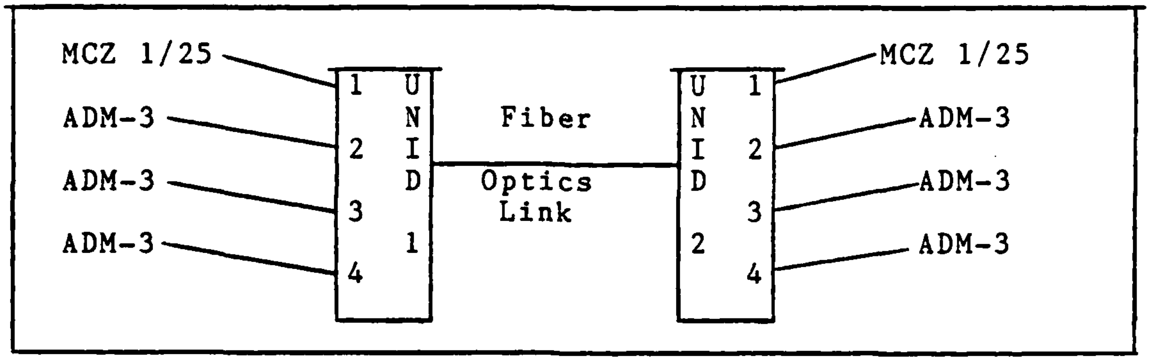


Figure 8. Prototype DELNET Terminal Configuration (Ref 21)

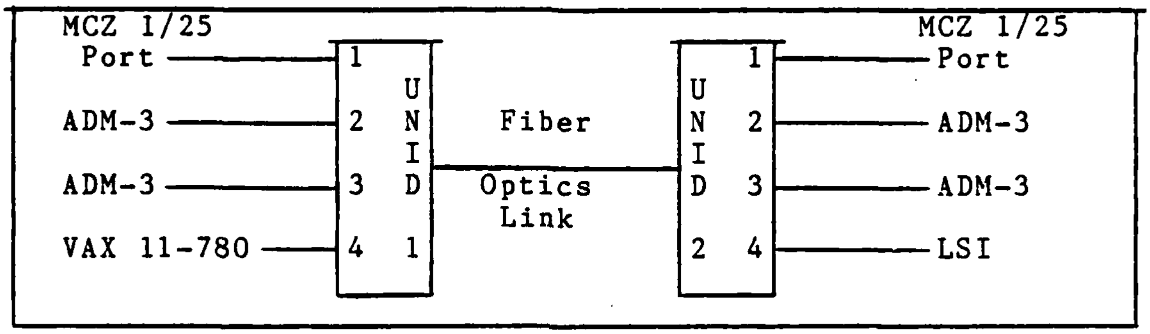


Figure 9. Prototype DELNET Computer Configuration (Ref 21)

in 1981 there was a limited demonstration of the DELNET using the two UNIDs configured as show in figure 8 and figure 9. The first test used only terminals and the second used both computers and terminals.

Protocols

Protocols are the rules governing the timing and formatting of data which allow one machine (or person) to communicate with another. In establishing the protocol for the DELNET, various protocols recommended by both national and international standards organizations were reviewed and those which seemed to best meet the stated requirements were

selected. A packet-switching protocol using the X.25 standard established by the International Consultative Committee on Telephones and Telegraphs (CCITT) was chosen (Ref 11:45).

Since the work will be done in stages and maximum flexibility is a design goal, a recommendation was made that the Reference Model of Open Systems Inconnection (OSI) developed by the International Standards Organization (ISO) be used for the DELNET. The development of specific protocols for the UNID and DELNET was started in 1982 (Ref 11) and they are being completed and refined in a concurrent thesis project (Ref 23). Since the UNID II will be a node in the network and it should support the three lowest layers of the ISO model, a brief description of the model and of the X.25 standard follows.

<u>ISO Reference Model.</u> The ISO model is intended to be a vehicle for the development of specific standard protocols within seven layers, shown in Figure 10. The layering divides a very complex task into smaller pieces, with each being relatively independent of the others. Several organizations have established, or are working on, standards which will operate in the framework of the ISO model.

At any given layer, a program communicates with the corresponding layer in another host or node. While the two hosts logically communicate directly with each other (the dotted lines in Figure 10), in fact all communications must pass through the lowest layer since the only physical

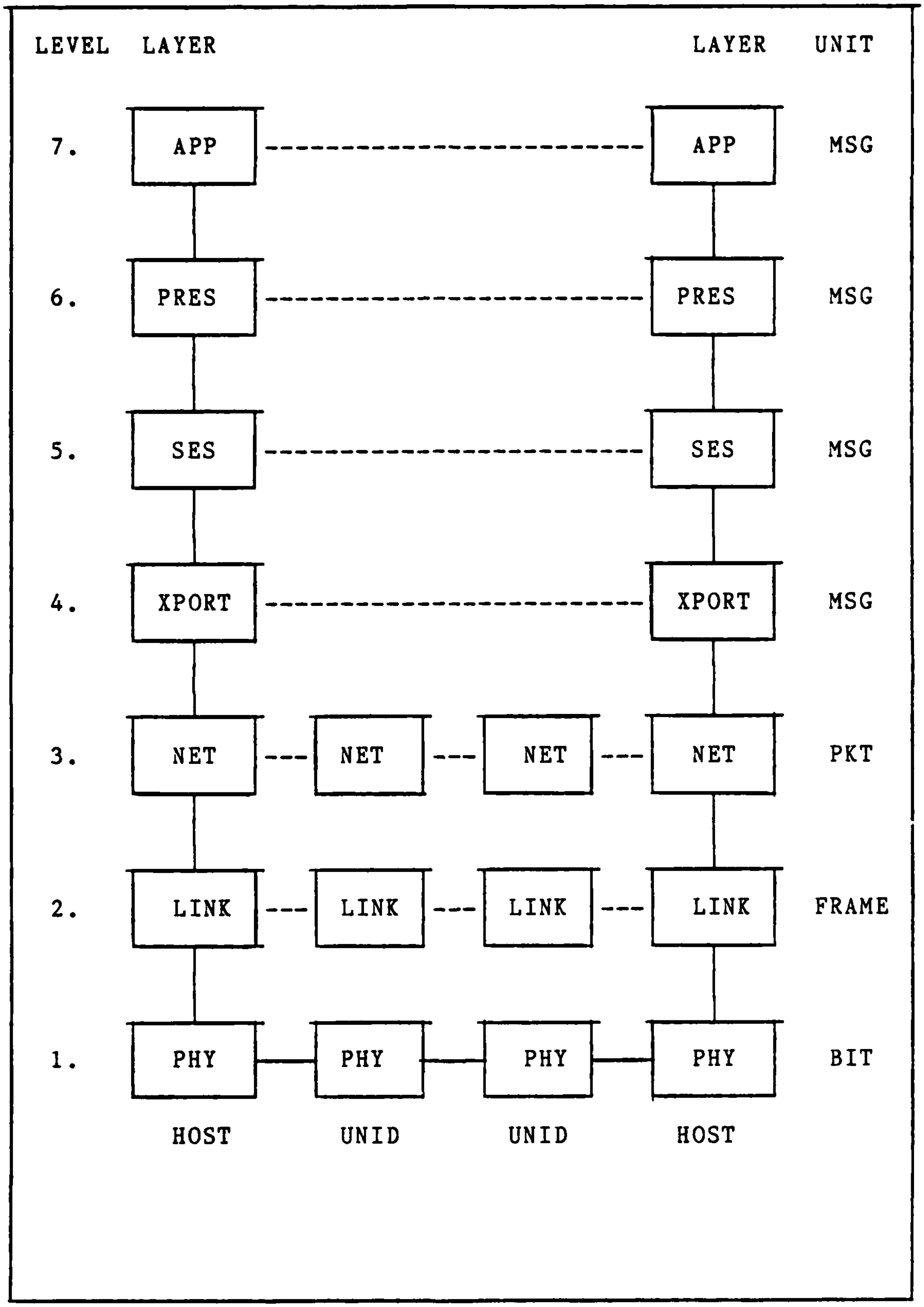


Figure 10. ISO Protocol Model with UNID (Ref 9:13)

connection is there (the solid line in Figure 10). The content of the applications layer is determined by the user and his requirements. This is the layer which supplies the user interface to the local network. The presentation layer and all lower layers provide support to the applications layer (Ref 30:430).

The presentation layer handles the data format transformations which can include data compression, file translation, end-to-end encryption, and virtual terminal protocols. Next, the session layer normally performs addressing and connection management of the network for the host, but in fact, some of these functions can be subsumed in the transport or presentation layers in an actual implementation (Ref 28:394-397). Once the data is formatted and addressed, it must be sent. The transport layer provides the host level communications facilities. This layer provides error-free end to end communications between hosts. Some examples of services at this level include error checking and recovery, flow control for the host, and establishment/breaking of a host to host connection.

Subnet. The lowest three layers make up the subnet. These layers route data through the network from one host to another while the higher levels are only concerned with the dialogue between the communicating hosts (end to end communications). In our application the UNID contains the subnet protocols. The access protocol selected for the

subnet is the X.25 recommendation by the CCITT. The CCITT recommendation X.25 describes the interface and procedures for packet switched service and it is defined in three independent architectural levels which can be used for the three subnet layers in the ISO model (Ref 8). Each of the subnet layers is discussed in some detail because they encompass the software and hardware functions of the UNID.

The network layer, refered to as the packet level by the CCITT, is the top level of the subnet and is responsible for routing and flow control. It determines the path a message, or packet, should take through the network from the originating host to the destination host. In most networks the two host computers may be separated by many nodes which are not directly involved (they only store and forward the packet) in the particular communication and there may be more than one path from host one to host two. Each node in the network must determine which way to send the data so that it will reach the intended destination.

In the 1980 revision of X.25, a number of significant technical enhancements were made at this level. Two of the most important were; the addition of provisions for datagram service, and the addition of a fast select facility to the virtual call service (Ref 8:2).

Datagrams are self-contained packets which contain sufficient address information to be routed to their destinations and they may contain up to 128 bytes of user data. No set up calls are required since it is a complete

message and not considered part of a larger unit. The fast select facility provision allows a full 128 bytes of user data to be exchanged during the call set up and clearing procedures for a virtual call (Ref 8:2). A more detailed description of these services can be found in the literature (Ref 26, 8, and 23).

The network layer must also deal with congestion. The network can become overloaded if the hosts initiate data into the network faster than it can be processed and delivered. Some method of controlling the amount of data in the network has to be used. One of the more common methods is the exchange of flow control messages with the network layers of other nodes. These messages can include information such as acknowledgement of receipt of data (ACK), the number of free message buffers, or the fact that the node is unable to receive data at the present time.

Messages from the transport layer may have a priority, it is the job of the network layer to insure that higher proirity messages are handled first. It must also deliver high priority messages to the transport layer before lower priority messages. This can be very important in military communications systems with a hierarchy of priorities.

Data Link Layer. This layer combines groups of bits into logical units called frames. It is the function of the data link layer to create, recognize, and control the flow of the logical bits transfered to/from the physical layer. The bit oriented link access control procedure specified in X.25 is the Link Access Procedure B (LAPB) which is equivalent to

Flag	Address	Control	Infor- mation	FCS*	Flag
F 01111110	A 8-Bits	C 8-Bits	I N-Bits	FCS 16-Bits	F 01111110

*Note: FCS = Frame Checking Sequence

Figure 11. X.25 Frame Structure (Ref 7:A8)

the ISO High Data Link Control (HLDC) standard (Ref 8:2). The frame format specified by the standard is shown in Figure 11.

Since the data link layer can receive bad data from the physical layer, a checksum (a calculated value based on each information unit in a frame) is added to each frame as it is sent. This checksum is compared with a locally generated checksum at the receiving node. If the checksums match then it is assumed that the received frame is correct, otherwise the transmitter must be notified that incorrect data is suspected and the frame should be retransmitted. The X.25 checksum is a 16-bit value based on the CCITT Cyclic Redundancy Code (CRC) generating polynomial with the dividend set to "1"s.

Physical Layer. This layer is the lowest level in a network and it provides the physical, electrical, mechanical, functional, and procedural services to define the physical connection between network equipment. The physical interface standard referenced by the CCITT for the host computer to network node is the X.21 digital interface standard. The internal protocols between network nodes are

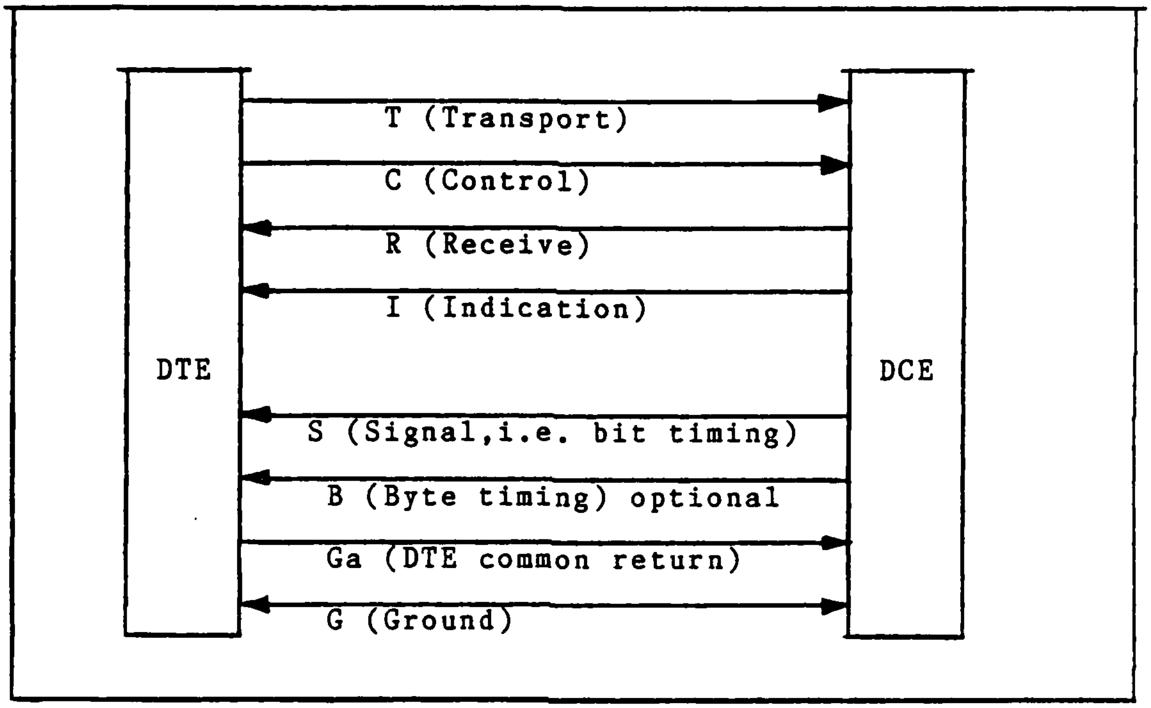


Figure 12. DTE/DCE Interface Connection in X.25
(Ref 26:109)

not defined by CCITT for the X.25 standard. The host computer or user device is defined as the data terminal equipment (DTE) and the network node (UNID in our case) is considered data circuit terminating equipment (DCE).

Figure 12 shows the eight lines defined by X.21 for the connector at the DTE/DCE interface. The X.21 standard is designed for the transmission of logical bits, so the "S" line provides a clock timing signal to define the bit boundaries. An optional "B" line allows for a timing pulse every eighth bit for byte allignment. The transport (T) and receive (R) are used for the data and/or signalling information. The control (C) and indication (I) are control type signals for handshaking information. The X.21

standard specifies a 15 pin connector but not all of the pins are used.

Standards

All new data communication equipment procured by the Federal Government shall conform to Federal Standard 1031 which was adopted from the Electrial Industries Association (EIA) Recommended Standard (RS) number 449 (Ref 6:72). This standard includes both the RS-422A and RS-423A electrical specifications and functional characteristics which define the DTE/DCE interface. The network ports on the original UNIDs are configured for RS-422 and UNID II must interface on this link so the two network ports will use RS-422. Most of the computers and terminals in the AFIT laboratories have only RS-232 interfaces, so the local to UNID II interface will use the RS-232C signals and hardware.

There is no U.S. standard which is equivalent to the electrical, mechanical, and functional characteristics of X.21, but RS-232C and RS-449 are essentially equivalent to the procedural characteristics of X.21bis (Ref 2:437). X.21bis is the analog counterpart to X.21 and is to be used for interfacing analog circuits until digital networks become widely available (Ref 26:238). The appendix of RS-449 contains a mapping of the RS-449 functional circuits to the X.21 functions (Ref 6). The actual RS-232 and RS-449 signals used on the DELNET are listed in Appendix B.

Summary of Requirements

This chapter summarized the requirements for the UNID II and the DELNET. The OSI model and X.25 access protocol standard were introduced and their correlation to the UNID II functions was briefly explained. The input requirements (Table I) and the Data Flow Diagrams (Appendix A) of the functional requirements model form the basis for the design and construction of the prototype UNID II described in the following chapters.

III. Hardware Design/Construction and Software Development

This chapter describes the design, construction, and software development for the UNID II during this continued development stage. The local subsystem monitor program and software to handle the local functions is covered first. Next, there is a section on the hardware additions to the local I/O card and the chip programming proceedures. The network card was completely redesigned because of the difficulty in supporting the required network functions with the previous design. The reasons for each change are discussed and the new design is developed in detail.

Local Module

The selection of the two modules to be used and the initial design of the UNID II was done in 1981 (Ref 10). The iSBC 86/12 used for the local processor was also purchased. The network wirewrap card was started in 1982 (Ref 21). It was not completed because the Signetics 2652 Milti Protocol Communications Controller (MPCC) chips were not received. A local I/O card was designed and built to add four serial channels for local hosts (Ref 3). The serial I/O card uses the parallel port on the iSBC 86/12 to pass all signals and data except power, ground, and interrupts which use the multibus connector.

Monitor. The iSBC 86/12 board has sockets for EPROM but all four were empty. The monitor program which is needed to load programs and work with memory and registers was not

provided or was erased. All testing and debugging of the UNID II has been done using the ICE-86 in circuit emulator on the Intel model III development system. The UNID II must have the ability to work as a stand alone device. This means that programs must be downloaded through the serial port on the iSBC 86/12 board and a monitor in EPROM is required.

The source code, in PL/M-86, for an iSBC 86/12 monitor was available both in print and on a floppy disk. The code had been copied from the source listing by a previous investigator (Ref 21) attempting to get it working. The source code was compiled, linked, loaded, and tested (in low memory) using the ICE-86 attached to the iSBC 86/12. The monitor operated correctly in low RAM so the code was relocated to reside in high memory at the EPROM addresses.

A HEX listing of the monitor program was prepared and using the EPROM burner on the S-100 system an attempt was made to load the program into four 2716 EPROMs. The program for the EPROM burner would not accept the HEX file. A check of the file revealed that it was not the Intel HEX format expected. Further investigation revealed that the files produced by OH-86 are in Intel extended HEX format to handle the additional addressing available with the 8086 microprocessor.

A program was written, using the Computer Inovations (CI) C-86 compiler, to convert the extended HEX file to a standard HEX file (Appendix D). This program produces an address that is less than 64K and recalculates the checksum.

The program does not do the entire translation because no check is made for boundaries between EPROMs. Two or more files would be needed to handle this case. The operator must devide the output file as necessary and correct the lines of code which cross boundaries. An end of file line must be added to each file produced and the offset information (a single line of code at the end) removed by the operator.

A working monitor on the iSBC 86/12 allows programs to be loaded but the host computer must also have a program to screen the commands and send files to the iSBC 86/12. The UNID II has no disk drives or other archival storage so programs must be entered into memory by the monitor using the serial or parallel port. The host computer acts as a terminal and a mass storage device for the UNID II.

The program in the host computer needs to accept keyboard input of commands and display them (echo) on the screen. It must output to the UNID II the commands and then display the data returned from the UNID. It needs to strip off the filename when a file is to be downloaded and use this information to get the file on disk and send it to the UNID. A program to accomplish this was written in C-86 for a CompuPro S-100 system using the 8088 and then translated to BDS C for the 8085 on the same system. The C-86 source listing is shown in Appendix C.

<u>Local Serial Card</u>. A local serial card was built (Ref 3), but the channels did not have drivers or line receivers. Completion of this card was the next step. The outputs were

designed to interface with RS-232 devices with jumpers on each channel to allow reconfiguration as a DCE or DTE channel. Motorola 1488 quadruple line drivers and 1489 quadruple line receivers were used for the level conversion. A revised schematic with the new circuitry was produced and the new circuits were added.

Local I/O Programming. The local cards of UNID II contain a large number of integrated circuit chips which have programmable options. Not only must each chip be programmed correctly, but the order in which they are set up can be critical! An extensive understanding of the programming method is needed if updates are to be made. While the required information needed to program each chip is available in the commercial literature (ref 12), finding and interpreting the data can be time consuming and difficult. Specific information on programming the chips, both because of the way the hardware was put together and software design decisions, is given in this section.

The serial input/output card on the local side is accessed through the parallel port on the iSBC 86/12 processor card. The data and control signals for the serial I/O card pass through a 50 conductor cable. Multibus connectors are used for ground, voltages, and interupts. The 8255A Programmable Peripheral Interface (PPI) on the iSBC 86/12 must be programmed before any of the chips on the serial I/O card can be accessed for programming or read/write operations.

Channel A of the PPI is used for data out of the serial card and channel B for data into the card. Both port A and B connect to a common data bus on the serial card but they are latched through bus drivers so that only one is active at a time. Control signals are put on channel C. Information written to Port C of the PPI remains on the control lines of the serial I/O card until cleared by writing an all zero word to the port. The PPI must be set to mode zero, port A in, ports B and C out using control word number eight (ref 12:8-91). Once the PPI is programmed, data and control for the serial card can be sent to the correct channel. It takes at least three instructions to place a single byte of data into one of the chips on the serial card. First the data must be output to channel B of the PPI and then a control word put on channel C. When the write is complete the control word must be zeroed by writing all zeroes to port C. In some cases it is also necessary to clear the data from port B which will require still another output instruction.

The serial card has two 8253 Programmable Interval Timers (PITs) used to provide clock to the four 8251 Programmable Communication Interface (called a USART in this document) chips. Each of the PITs has three independently programmable counters. PIT one is used for the USART on channels one, two and three while PIT two provides clock to channel four. Three characters are required to program each channel to be used. The first character defines the mode of operation and the remaining two determine the frequency of the output.

Programming information for the PIT and all the chips on the serial card is provided in appendix E.

The USARTs are not reset automatically on power up so a reset instruction must be sent prior to filling the mode registers. The USART reset instruction must be placed on channel C of the PPI for at least 2500 micro seconds to insure a reset. The reset instruction will reset all four USARTs so a new mode setting must be written to each before it can be used again. Normally each USART will have different settings to match a local computer or terminal but for testing all four will operate asyncronously at 9600 baud. A side affect of having all four the same is that less writes are required to program the chips since the data on channel B only has to be written once.

Local Software. The local I/O subsystem needed to be tested and operational software was required for the lower three levels on the ISO model. Programs for the Z-80 UNIDs was already written and tested so a decision was made to translate the source code from PLZ to PLM-86 for use on the UNID II. Doing this would insure a high probability that the different UNIDs would operate together in a network since the same logic and software design procedures would be used on both. Additionally, since many test points were built into the original software, it would be easier to follow during the debug phase. With this software a complete test of the local subsystem would be possible.

The idea that PLZ and PLM-86 were slightly different

dialects of the same language was disproved very quickly. The differences in the way constants are declared, the nesting of if statements, passing of parameters, and bit manipulation required some interesting, and at times frustrating conversions. Another difference was that the Z-80 UNID has interrupts on the receive channels only, while the UNID II uses both transmit and receive interrupts. Also, the port drivers and a few low level routines were written in assembly language for the Z-80, while the 8086 version is written completely in PLM-86.

As the source code was converted, a source file was built and stored on a floppy disk using the Intel MDS. The Intel MDS was used for all PLM-86 software development since the only available compiler runs under the ISIS-II operating system which we presently have running only on the Intel systems. The majority of the errors were in entering the code correctly but there were some PLM-86 constructs which were incorrect and had to be changed. This translation took longer than expected because the languages were so different that a through understanding of the original source code and the intent of each operation to insure that the translation would accomplish the same task as the original.

Network module

The network card was considered to be complete except for the replacement of the 8251 USART with a 2652 MPCC and the addition of a second 2652 MPCC (Ref 21,69-70). The attempt

to complete this card uncovered difficulties in several areas. Each of the problem areas is covered and the method of solution discussed.

A comparison of the UNID II schematics and documentation with the planned DELNET protocols uncovered a discrepancy. The UNID II was designed to use the CCITT X.25 protocol while the DELNET using the original UNIDs is configured to use the EIA RS-422 protocol. The Signetics 2652 MPCC does not generate or use several signals designated for use in the DELNET. One solution would have been to build control circuits to handle the control lines but board space was limited and it is difficult to forsee all the possibilities.

A second, and prefered, solution was to find a chip that would do everythig the 2652 MPCC could do but also had the necessary handshaking signals. Available documentation (Ref 7:5-267,5-296) indicated that the Fairchild 6856/3846 Synchronous Protocol Communications Controller (SPCC) would work. The 6856/3846 SPCC supports the same protocols as the 2652 MPCC, has both byte and word (16 bit) data lengths, and it produces and uses the Data Terminal Ready (DTR), Request to Send (RS), and Data Set Ready (DSR) signals. The only area which it does not match the 2652 MPCC is in speed. The 2652 MPCC currently has versions that run at 2MHz while the 6856/3846 SPCC has a maximum speed of 1MHz. The speed factor does not seem critical at this stage in the development and higher speed versions will be available later.

An added advantage of using the 6856/3846 SPCC is that

they do not require the special handling that the 2652 MPCC needed. The chip select signal is an active low and the 2652 MPCC needs a high signal but the 6856/3846 SPCC uses the signal provided directly without inversion. If the 3846 SPCC version is used the control and timing signal are designed to work on an Intel 8080 system. The UNID II design had additional circuitry to create the Enable (E) and Read/Write (R/W) signals. The 3846 SPCC replaces the E lead with a Read Pulse (RD) and the R/W with the Write Pulse (WR), both signals are easily obtained on the board. For these reasons the 3846 SPCC will be used for the network links, one chip on each of the two full duplex channels.

An attempt to install the new chip for the serial network link presented new problems. The circuit diagram did not show this chip or the connections required for installation even though a space was reserved for it on the card. The installation was complicated by the fact that it is necessary to know how the 8089 IOP will be programmed before the signals can be routed. A review of the expected function of the two network links was necessary before a decision could be made.

The two network links are bi-directional (full duplex) channels and traffic can flow in both directions from the UNID II. This is important because it implies that both receive links should be serviced on an interrupt basis. Also it was necessary to know if the transmit sides would be polled or interrut driven. The answer was provided by Palmer

(Ref 21:66-67):

> The receiver Data Available (RxDA) signal is routed to one of the 8089 channel's DRQ input for DMA source synchronization, and the Receiver Status Available (RxSA) is connected to the IOP channel's respective EXT input to provide an external termination condition. The RxDA signal is activated when data is available in the MPCC's receiver buffer, and RxSA is activated when an end of message is received. For DMA destination synchronization of the IOP during transmission, the Transmitter Buffer Empty (TxBE) signal is to be used as a condition for the synchronization of the IOP channel's DRQ input.

The 8089 IOP has two DRQ inputs and a single serial link needs both of them. Since two IOPs are needed and only one is available, an alternative was needed. If the IOP does word transfers, instead of string transfers, the 8259A Programmable Interrupt Controller (PIC) can be used to handle the interrupt requests. The trade off is a very large increase in CPU overhead since the IOP control registers must be reloaded on every input or output word transfer. The result is a slower operation with the IOP than if there was no IOP involved. This is not an ideal situation.

With the decision made to use the PIC for interrupts and do word transfers, it was necessary to decide how to get the executable code into the RAM on the network card. The only monitor routine is on the local processor card and the functions available are limited. The situation is complicated by the fact that the dual ported memory is addressed at low memory, 0 to 7FFF hex, on the local side and at high memory, F8000 to FFFFF hex, for the network card. Programs could be loaded into the shared memory but

the network card could not execute the programs because of the incompatable addresses. The addresses on the network side could not be changed without adding a ROM, with a start up procedure, to the network card.

A re-evaluation of the entire network card was done to see if the problems could be corrected. Several methods were considered. Initially, the addition of ROM with the network programs burned in was reviewed. To do this, additional decoding circuitry would have to be added and the addressing for the network card local memory would have to be changed from low to the top of memory. This would still require the word at a time transfer of frames with the associated overhead.

A second idea was to add a second 8089 to the board. This can be done using the second request/grant (RG/GT1) line from the CPU. This approach would still require the ROM with the network program, but would reduce the overhead on the CPU and the IOPs could be used as intended. One drawback is that the IOP attached to RG/GT1 is always the lowest priority and traffic to this link could be delayed if both links are being used at 1Mbps.

The final idea was to remove the 8086 and make the network card a remote cluster of 8089s. In this case the system address space is shared and the local addresses on the network card are in the I/O space of the system. This method would allow the best utilization of the 8089s since in the local mode (presently used on the network card)

parallel operation of the processors (8086 and 8089) is limited to cases in which the CPU has instructions in its queue that can be executed without using the bus (Ref 14:7-6).

The reason for having a CPU on the network card was to reduce the processing load on the local CPU. A review of the software for the Z-80 UNID (Ref 23) showed that very little processing was done on the network side and the processing was only to determine the destination of packets received. It does not appear that the CPU on the network card is necessary and there is not enough room to add the components required for the first two alternatives. For these reasons and also because the 8089 IOP would not be used as it was intended except in the third configuration, the remote cluster of IOPs was selected.

Network Card Design

The redesign of the network card to make it a remote cluster of 8089 IOPs instead of the local configuration with an 8086 CPU and an 8089 IOP requires some major modifications. The new design is more efficient and does not increase the chip count on the board. Changes will affect the 8086 CPU, 8288 Bus Controller, 8289 Bus Arbiter, 8259A PIC, address decoding circuits, and discrete gates that produce signals needed only in the local mode. Each of the changes will be discussed in detail.

A comparison of the signals at the pins of the 8086 and

GND	1		40	Vcc	GND	1		40	Vcc
AD14	2		39	AD15	AD14	2		39	AD15
AD13	3		38	A16/S3	AD13	3		38	A16/S3
AD12	4		37	A17/S4	AD12	4		37	A17/S4
AD11	5		36	A18/S5	AD11	5		36	A18/S5
AD10	6		35	A19/S6	AD10	6		35	A19/S6
AD9	7		34	$\overline{BHE}$/S7	AD9	7		34	$\overline{BHE}$
AD8	8		33	MN/$\overline{MX}$	AD8	8		33	EXT1
AD7	9		32	$\overline{RD}$	AD7	9		32	EXT2
AD6	10	8086	31	$\overline{RQ}/\overline{GT0}$	AD6	10	8089	31	DRQ1
AD5	11		30	$\overline{RQ}/\overline{GT1}$	AD5	11		30	DRQ2
AD4	12		29	$\overline{LOCK}$	AD4	12		29	$\overline{LOCK}$
AD3	13		28	$\overline{S2}$	AD3	13		28	$\overline{S2}$
AD2	14		27	$\overline{S1}$	AD2	14		27	$\overline{S1}$
AD1	15		26	$\overline{S0}$	AD1	15		26	$\overline{S0}$
AD0	16		25	QS0	AD0	16		25	$\overline{RQ}/\overline{GT}$
NMI	17		24	QS1	SINTR1	17		24	SEL
INTR	18		23	$\overline{TEST}$	SINTR2	18		23	CA
CLK	19		22	READY	CLK	19		22	READY
GND	20		21	RESET	GND	20		21	RESET

Figure 13. 8086 and 8089 Pinouts (Ref 11:7-1,7-51)

the 8089 shows that only nine are different (Figure 13). The small number of pins to be rewired makes the use af the 8086 CPU socket for the second 8089 IOP practical. Of the nine pins that must be changed, three (EXT1,DRQ1, and DRQ2) will come from the 3846 SPCC, two will go to the 8259A PIC (SINTR1 and SINTR2), one comes from the address decode logic (CA), one from the multibus (SEL), one goes directly to the other 8089 ($\overline{RQ}/\overline{GT}$), and the remaining one is not used and is connected to ground.

The memory and I/O ports on the network card will be in the system I/O space. This is normal in the remote configuration of a multiprocessor 8086 system but was not the method used previously on the network card. The card

used two 7485 (4-Bit Magnitude Comparators) and some discrete gates to determine if the addresses were in low memory. If low memory was addressed, the local bus was enabled using one of the 8288s. Any addresses not in low memory caused the other 8288 to enable the system address lines. The remote cluster allows the removal of one of the 8288s and removal of the decode circuits for accesses to the network local memory and I/O devices.

The 8289 Bus Arbiter has some major changes in the routing of the signals that appear on its pins. The only CPU will be the one on the iSBC 86/12 and the control signals will come from the multibus instead of being generated and used locally. There are also some changes in which signals are tied to ground and Vcc. Figure 14 shows how the 8288 and 8289 are connected in the new configuration.

There are four interrupts that are generated on this card and the multibus has only eight interrupt lines. However, the iSBC 86/12 was designed so that slave interrupt controllers can be added to other boards on the bus and a single interrupt line used for eight additional interrupt sources. The important thing is that the cascade signals are placed on the bus lines AD8, AD9, and AD10 so they are available to any card that needs them. The 8259A PIC on the network card will be removed and the four SINTR signals will be connected to the multibus interrupt lines IR0-IR3. The 8289A PIC will be added to the local I/O card and used as a slave so that all eight interrupts generated on that card

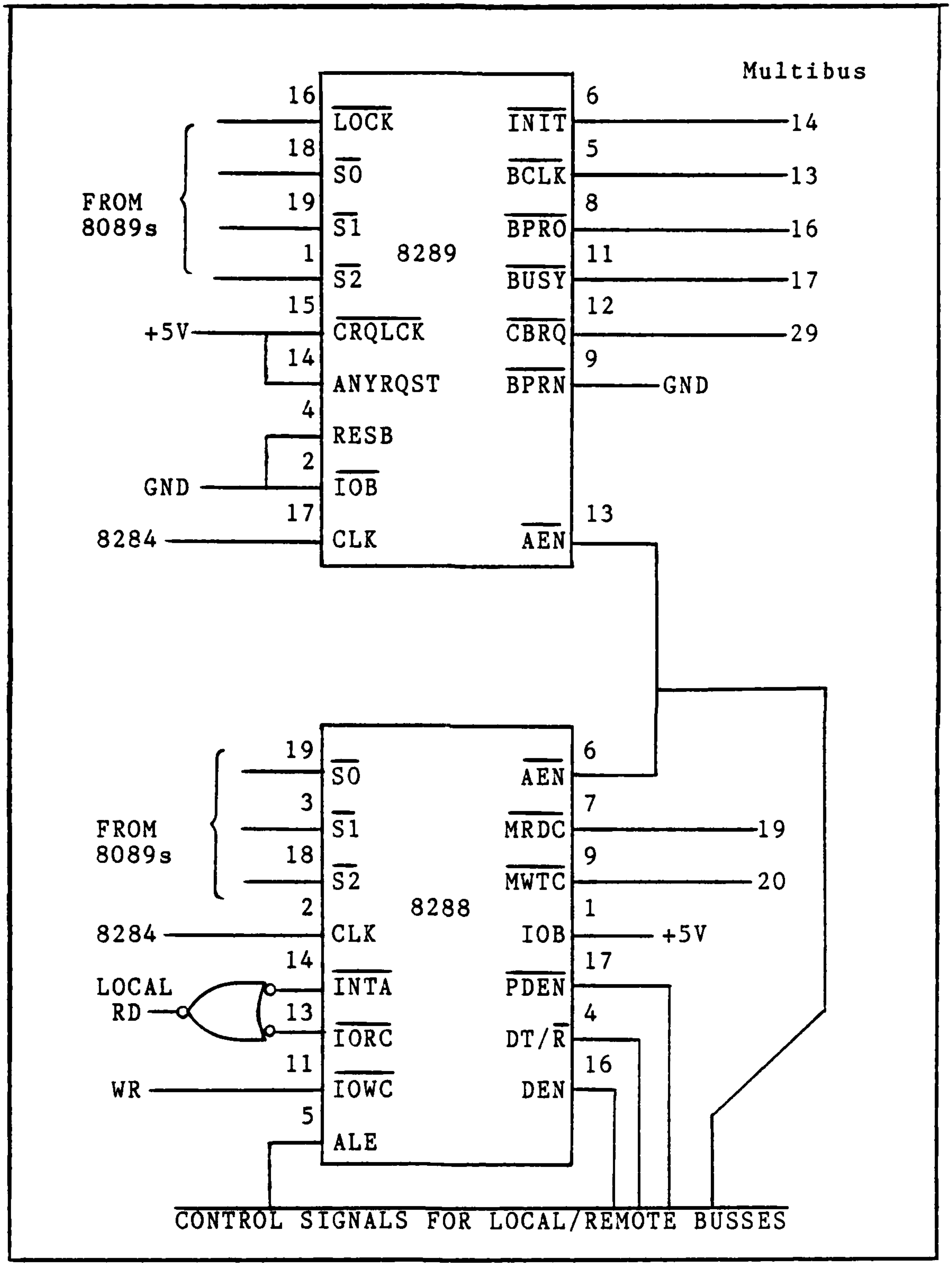


Figure 14. Network Card Bus Control Circuit

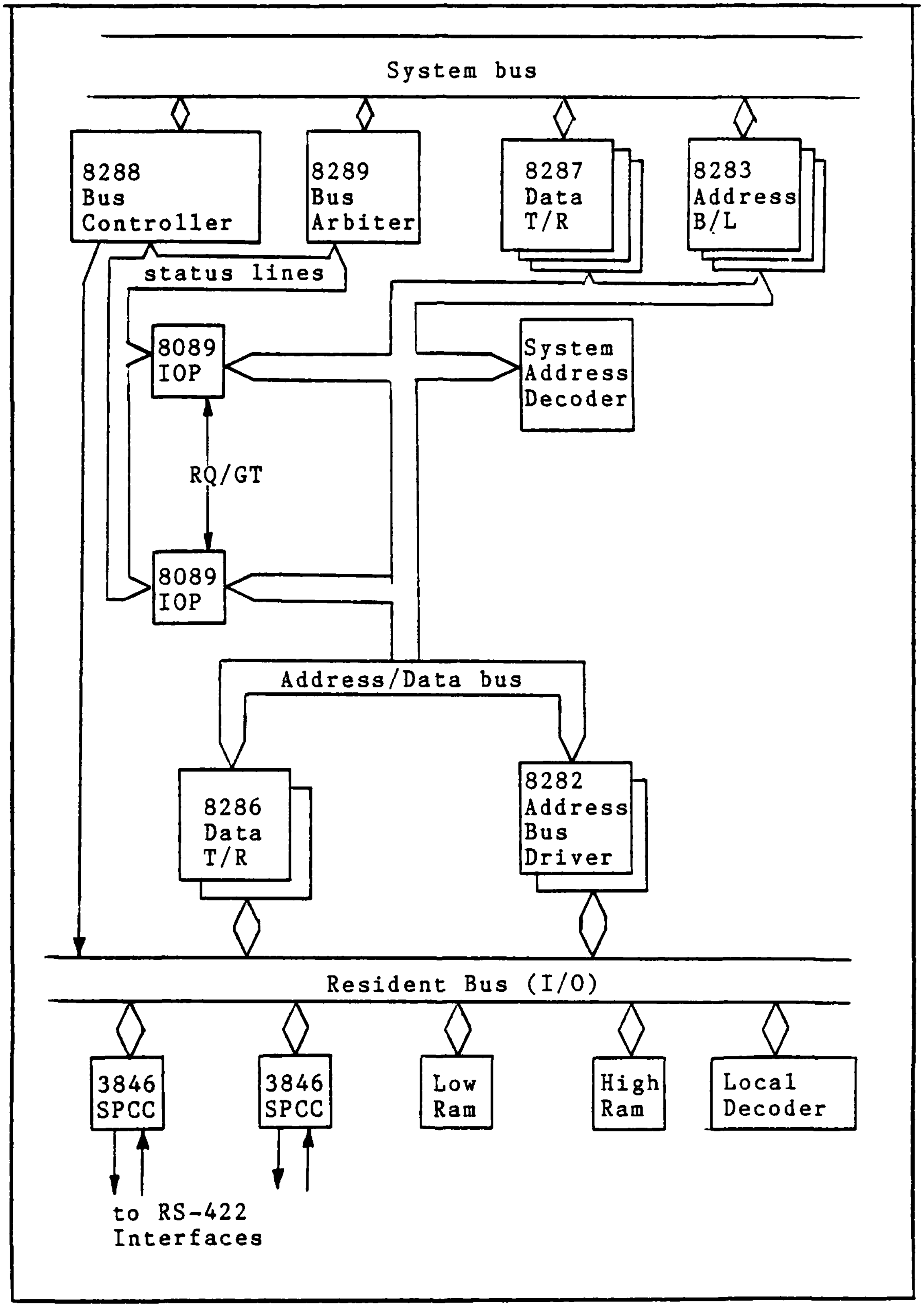


Figure 15. Block Diagram of Network Module

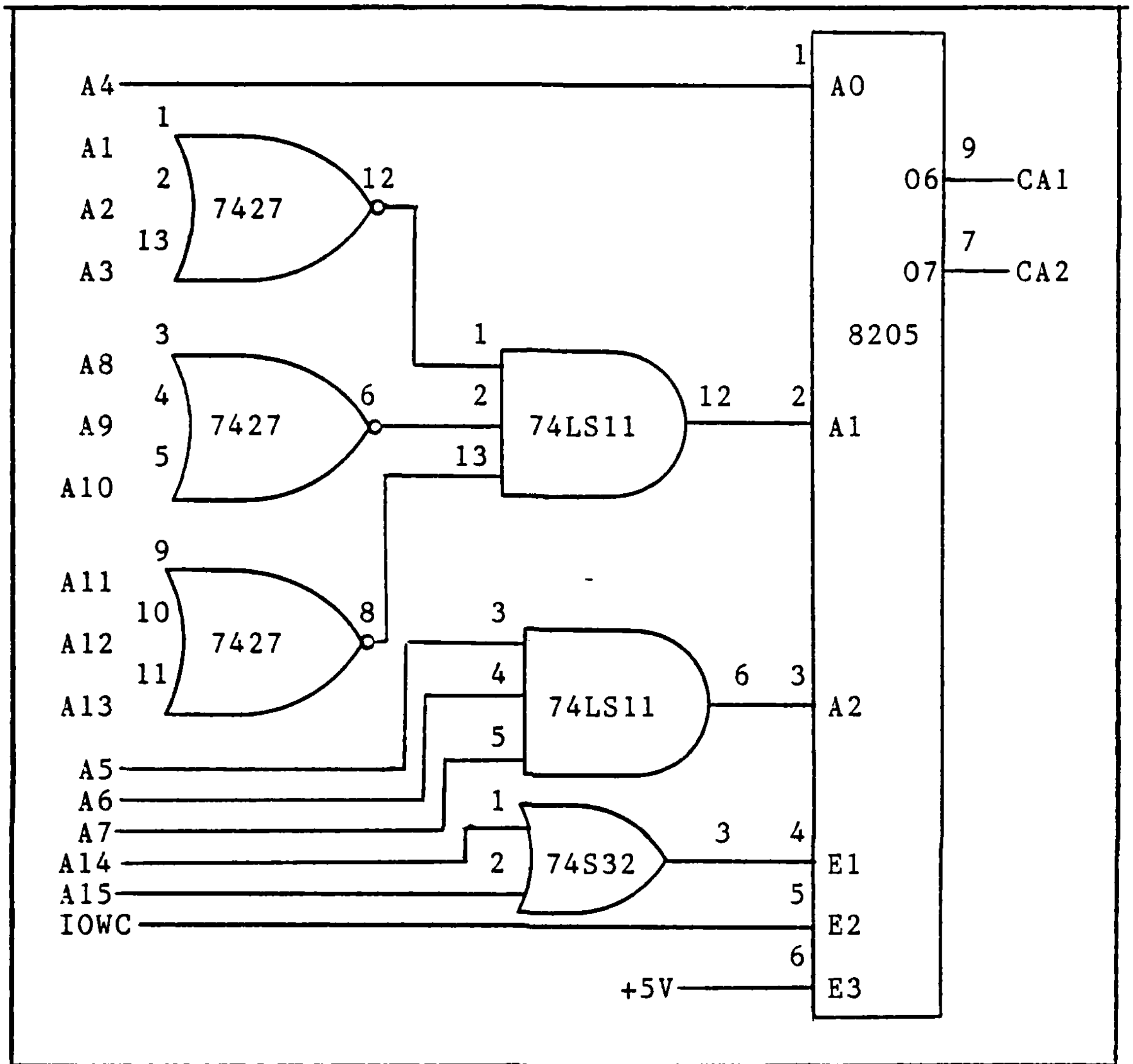


Figure 16. Decoder for 8089 CA from System Bus

can use a single multibus interrupt line instead of the seven it now uses. Figure 15 shows the major functional relationships on the network card once the changes are made.

The network card had a single 74S288 ROM to decode the address lines and generate the chip select signals. This method worked fine but the ROM did not produce a chip select for the second serial chip because it had not yet arrived. Also, because only four address lines were decoded (RA12-

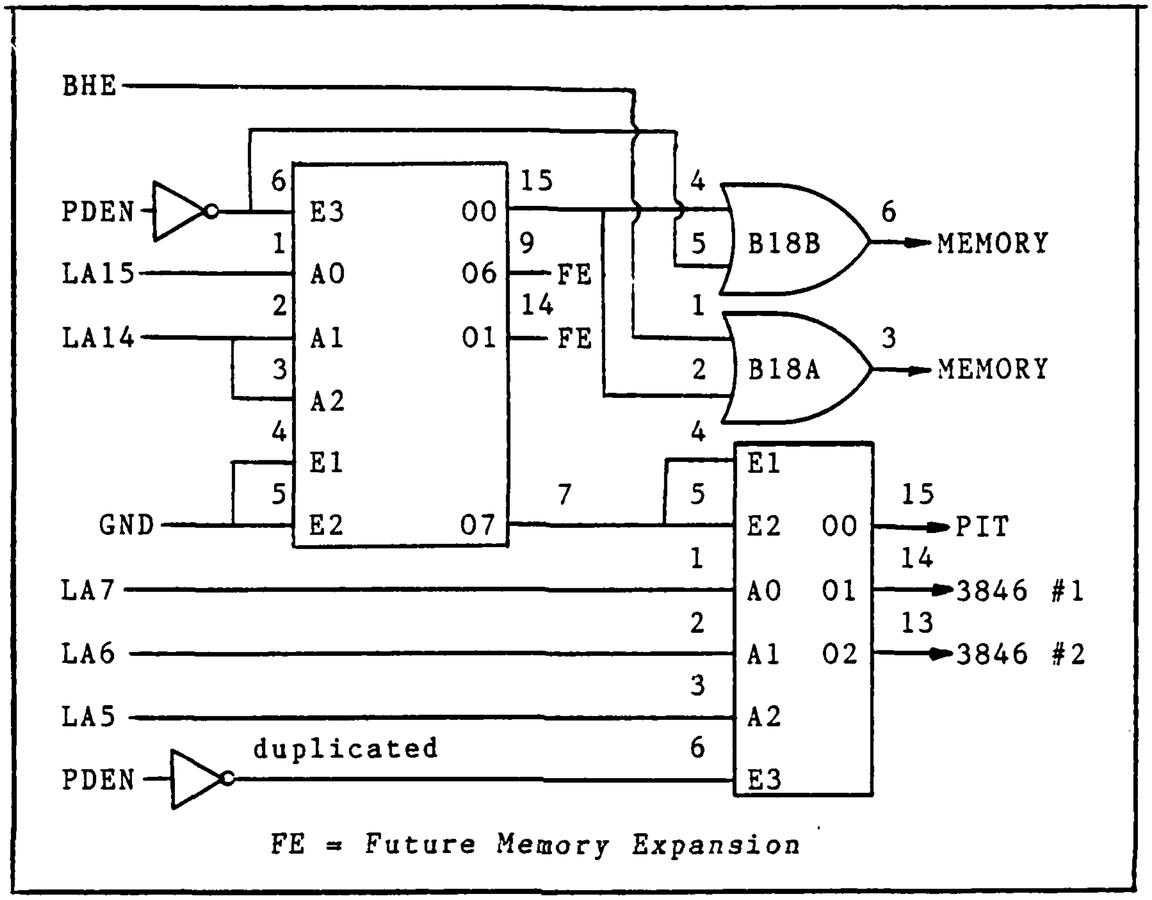


Figure 17. Local Bus Decode Circuit

RA15), the smallest address space assigned was limited to 4K bytes. This meant that each chip used a 4K byte space even if only 2 bytes were needed.

The change to a remote cluster meant that not only would the decode circuit for the local bus on the network card had to be redesigned but another one was needed to decode the system bus. The two 8089s appear as I/O devices to the iSBC 86/12 CPU and the Channel Attention (CA) of the 8089 is activated when the 8086 sends a write command the the address specified. A decode circuit for the two addresses is needed.

Addresses for the two 8089s were needed. They are in the 8086 address space and could not conflict with the addresses used on the iSBC 86/12. A review of the manual on the iSBC 86/12 showed that the highest address used was DE hex. It was decided that address E0 and F0 would be acceptable. The circuit to decode the add.ess lines and produce the CA signal for the 8089s is shown in figure 16.

The removal of the 8089 from the local bus leaves only three chips (two 3846 SPCCs and the PIT) plus the memory to be selected. Only one 8K word (16K byte) block of memory is presently installed but it is possible that more will be needed in the future. Three 8K word segments are reserved for memory, even though only one (addressed at 0-16K) is connected. The top 16K byte segment is used for I/O devices. The PIT is addressed at 0C00 hex, the first 3846 SPCC is at 0C20, and the second 3846 SPCC at 0C40. The circuit is shown in figure 17.

The 3846 SPCC has eight addresses to access the data and control registers. The decoder enables the chip select pin when the address is in the correct block assigned (20 hex in length) and the lower three local bus address lines connect directly to pins on the chip. If word (16-bit) transfers are made the least significant address line is not used, but if byte transfers are used then A0 selects the upper or lower half of the 16-bit register.

The SPCCs can be programmed to transmit and receive serial data in the format shown previously (Figure 11). They

automatically perform zero insertion and deletion to prevent any bit sequence in the frame from being misinterpreted as a flag, an abort, or an idle. After the beginning flag is transmitted, a zero is inserted into the serial data after five consecutive ones have been transmitted. Also, when receiving, the number of consecutive ones are counted. If five ones in a row are received, the sixth bit is discarded if it is a zero (Ref 7:5-271).

The SPCCs are completely contained in the network card local bus area. The sixteen data lines and three address lines come directly from the local bus drivers. Four signals connect directly to the associated 8089 and six are passed through level conversion circuits. The serial channel signals are interfaced to the network through RS-422A compatable line receivers and line transmitters (Motorola MC3486 line receivers and MC3487 line drivers are used on the Z-80 UNIDs).

The clock is provided by the PIT. The circuit is designed so that either local or remote clock can be used. The network will operate much better if the receive clock is provided by the transmitter along with the data. This insures that the receive circuit is operating at exactly the same clock rate as the transmitter. The RS-422A specification provides for the clock signals but the Z-80 UNIDs did not implement it. There are plans to change the UNIDs and the switch will allow either method to be supported. The connections to one of the SPCC chips

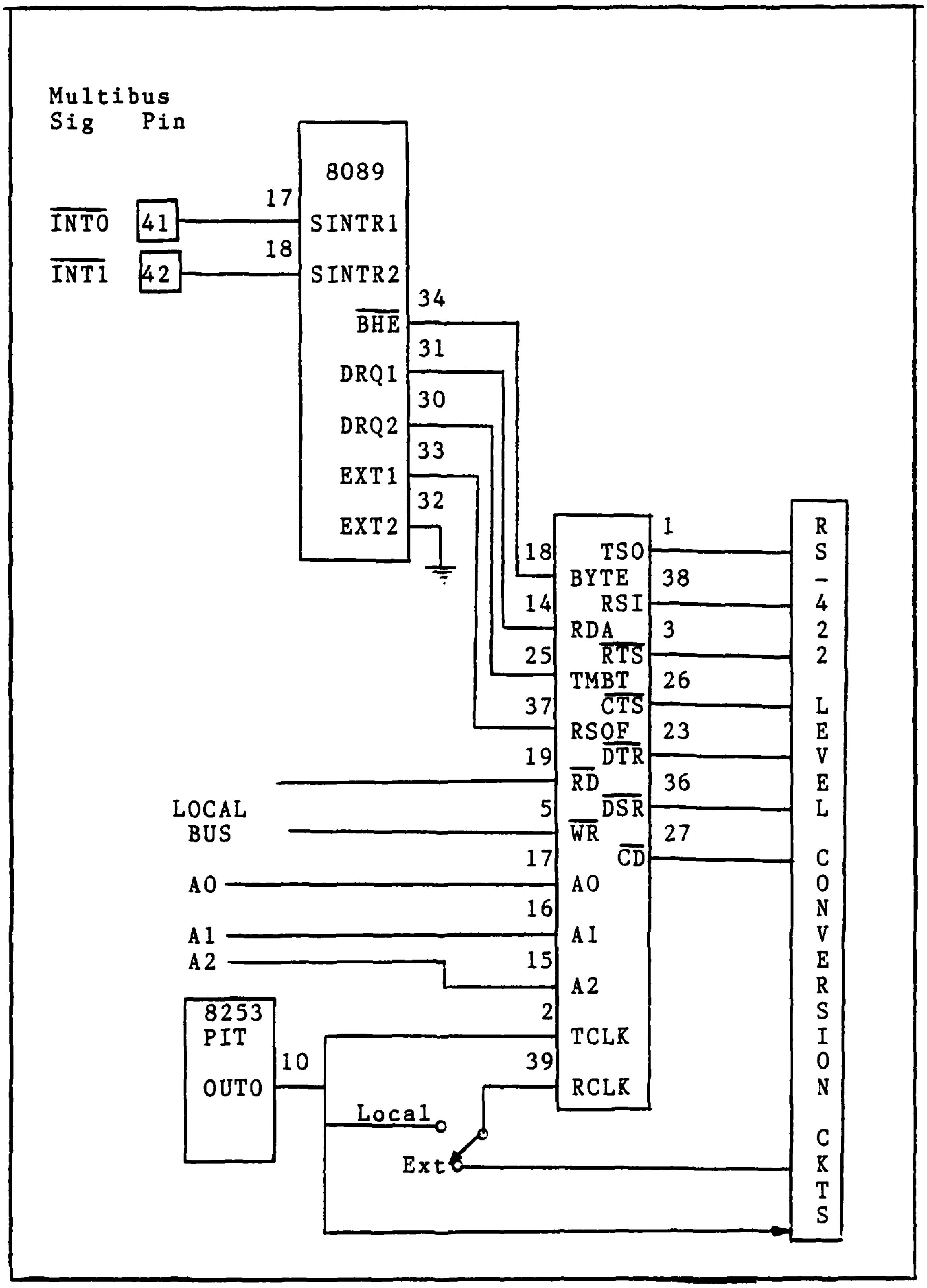


Figure 18. Network Serial Channel (one of two)

(excluding the data lines) is shown in figure 18.

Summary of the Network Module Design and Construction

The UNID II local subsystem currently has the hardware and software to send and receive packets (to other local hosts) using the X.25 datagram protocol. A monitor program for the iSBC 86/12 is installed in EPROM to reduce the dependence on the ICE-86A. The UNID II can not operate as a stand alone system, but it can now work with any host system providing the interface software is set up for the host being used. The basic software is written in a high level language (in this case C) to make the installation on a new host easier.

The network card was redesigned to make it easier to use and to take advantage of the functions provided by the co-processors. The redesign caused a delay in the hardware development, but the software should be easier to develop and the overall operation should be better because the network module is now designed to use the 8089 IOP chips as they were intended and all interface are available in hardware. The network subsystem software was not written because of the redesign effort. The overall functional diagram of the UNID II as changed is shown in figure 7.

IV UNID II Test Procedures

This chapter presents the testing performed during this effort. Included is testing of the hardware built previously to insure that everything was still operating as expected. While both hardware and software testing was done. New work was done on software so the majority of the testing was software related. Testing was done on the Intel Series II MDS using the ICE-86A attached to the circuit cards and also with the iSBC 86/12 being downloaded from a host computer. The testing is presented in the sequence in which it was accomplished. Initially the previous work was tested with the programs developed by the original investigators, (Ref 3, 21) and available on 8-inch floppy disks. Next, was testing of the local I/O card including the hardware and software developed during this effort. A summary of the test procedures and test results conclude this chapter.

Initial Hardware Testing

When this effort started the UNID II consisted of three boards, a commercial Intel 86/12A, a wirewrapped local I/O card, and a wirewrapped network interface card. The two wirewrap boards were developed at different times by two independent investigators. There was a time lapse from when the boards were completed and the start of this effort so it was important to insure that nothing had gone wrong during the period they had been unused. The cards were tested starting with the local module and then the network card.

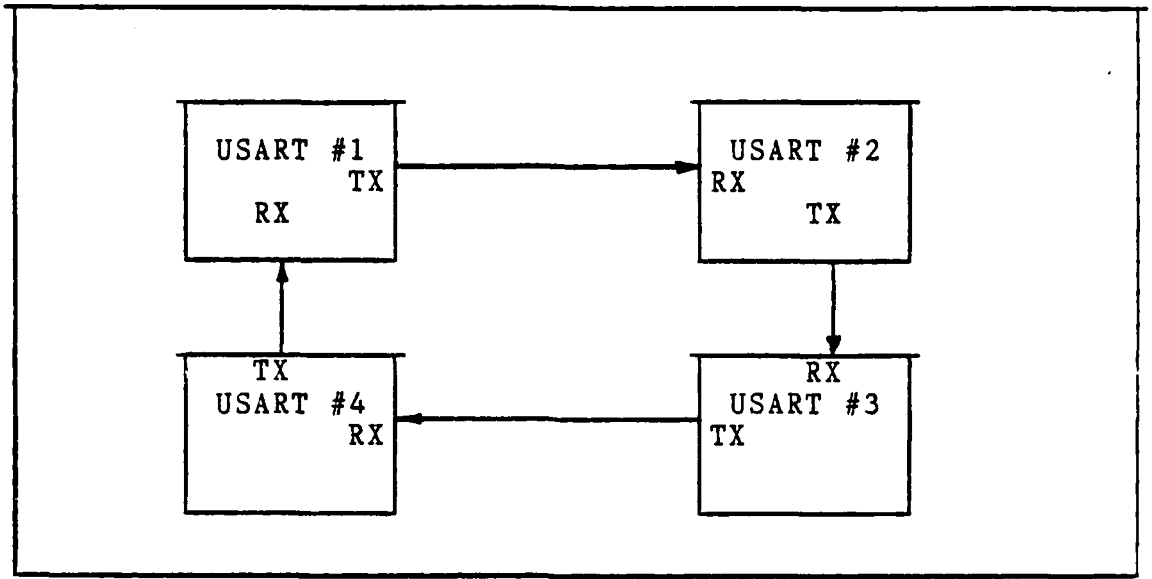


Figure 19. Local I/O Test Configuration

Local Subsystem. The 86/12A card was attached to the ICE-86A on the Intellec system and the "CARS" program from the ICE-86A manual (Ref 15) was run. This program is to familiarize users with the ICE-86A but it is also a good check of the memory and timing circuits on the 86/12A. The program executed completely with no problems except for those induced by the unfamiliar operator.

The local I/O card attaches to the parallel ports of the 86/12A card, so it was set up as soon as the 86/12A testing was successful. The "SCIB" program was developed to test the four I/O ports by sending messages from one port to another in a predefined sequence. The reason for the sequence is that the input and output pins on the USARTs were wired so that the transmit of number one went to the receive pin of number two and so on until number four transmit was tied to the receive of number one (Figure 19). No line drivers or

receivers were installed so it was not possible to interface to other computers or terminals.

The SCIB program has several important caveats which reduce its value as a test tool. First, it only works in the asynchronous mode and does not operate correctly in the synchronous mode (noted by the original investigator). When using the program on the ICE-86A it sometimes goes to undefined locatations for no reason and it must be stepped through before starting the run. Finally, it tests only one pair, one tramsmit channel and the corresponding receive channel, at a time with no real indication that the program is really doing what you asked it to do.

The SCIB program was loaded in the iSBC 86/12A and executed as instructed (Ref 3). The results obtained in these runs duplicated the results contained in the previous report. While this does not prove that everything is correct it does indicate that the I/O card is still operating as it did when the previous investigation ended. The entire local subsystem is operating correctly.

Network Subsystem. There are two programs provided for the network module. The first is a demonstration of the network module routines, and the second contains the TASK1 and TASK2 procedures for the 8089. The two programs work together to interface the network card to an ADM-3A monitor for this demonstration. The local module CPU was strapped in the halt mode and the int7 (RESET) was tied high for this testing. It was discovered that the two programs were

combined into a single program since no ASM-89 was available. The TASK1 and TASK2 blocks were entered as data rather that appearing a a seperate program.

The program N8612 was loaded into the shared memory on the local card using the ICE-86A connected to the network card. Running this program and entering the required responses at the ADM-3A keyboard tested the network module and its ability to access shared memory. No problems were found during this test sequence but it was later discovered that the upper addresses on the local address bus were grounded. This was done earlier because the 8282 chip had failed and none were available. All the hardware worked exactly as the previous investigators indicated it would.

Testing of New Work

The new work consisted of determining how the monitor program worked and what features it provided, a standard hex to extended hex conversion program, testing the burned EPROMs, hardware additions to the local I/O card, and the local software developed. The first three items all relate to the monitor program and the actual testing overlaps so it will be covered as a single subject.

Monitor. The monitor PL/M-86 source listing was compiled, linked, and located in low memory. The ICE-86A was used to load it in the memory on the iSBC 86/12A and start execution. An ADM-3A terminal attached to the serial port of the 86/12A was used as the I/O device. A "U" was typed and

the sign on message was displayed on the ADM-3A. Various letters and numbers were typed in and the response noted to see what commands worked and how they were called.

Since the monitor worked in low RAM a decision was made to relocate it to high memory locations and put it in EPROM. When it was discovered that the hex file produced was not usable on the EPROM burners available at AFIT, a conversion program was developed. The testing of this program was done in several stages. First, it was tested to insure that a correct checksum was generated. Next, a test was done to see if the newly calculated addresses were correct. A final test was done to insure that nothing was left out or added.

The conversion program has to change the address fields on some lines and discard other lines. Every time it calculates a new address the line must have the checksum corrected so that the EPROM programmer will accept the code. Some known good programs without the extended address fields were passed through this program and the output compared to the input file to insure that the checksums were unchanged. To be sure that the program was calculating the checksum, some of the input checksums were changed and the output checked to see if they had been corrected.

Testing the addresses calculated was a little more difficult. There was no good way to automatically test the code so a manual check was done at several locations in the output code. This was not as difficult as it seems since the addresses go in increments of 10 hex or less and the offset

remains the same for many lines. The addresses checked were correct.

A final visual check of a listing of the HEX file produced was made to insure that all the extended addresses were removed and to see if anything was obviously out of place. The end of file line was missing. It was decided to add this line using a text editor rather than modifying the code to handle this. At this point the program was burned into four 2716 EPROMs.

The EPROMs were inserted in the sockets on the iSBC 86/12A card, an ADM-3A was connected to the serial port, and the letter "U" was typed on the keyboard. Nothing happened. The EPROMs were not working but the reason was not known. At this point it was decided to use the ICE-86A to put another copy of the monitor in low RAM and use this to inspect the EPROM locations. This indicated that the jump instruction that should have been at location FFFF0 hex was not there, and one byte in the file had been overwritten. The ouer write was because the file was manually split and a mistake was made which put one byte too many in the upper file.

The file was corrected and run through the convertion program again and the resulting file was burned into the four EPROMs. Again there was no response when the monitor was used. The ICE-86A procedure was used again and it was found that the EPROMs were in the incorrect sockets, the high and low bytes were reversed. This was corrected and the monitor was used again with a correct response from the

monitor.

A program had to be written to interface a host computer to the monitor so that programs could be loaded into the iSBC 86/12. The important thing was to find out how the monitor reacted to a LOAD command. Every time the "L" was entered the monitor returned an error condition. A close check of the source code showed that there was a time-out that would drop the user out of the LOAD section if data was not entered fast enough. Typing the LOAD command followed by any key with the repeat key kept the timeout from occuring until the key was released. The loading commands were the only ones with a timeout. All the information needed to write an interface progran was now available.

The interface program was tested to see if the simple commands would be accepted and the correct response returned to the host, in this case the CompuPro 8085/88 dual processor. Initially there was no response from the iSBC 86/12A to any commands. The RS-232 cable connecting the two systems was found to have a pin in the wrong location. This was a cable made for this testing and the wire was incorrectly installed. The monitor now responded correctly to the simple commands, such as readouts of memory locations and changing register contents.

The interface software was tested to see if it would down load a HEX file to the iSBC 86/12A. The file was found and it was loaded into the memory of the UNID. However, no testing was done to see if the program would run once it had

been placed in memory. The reason for this was that there were no additional terminals available where the computer was located.

Input/Output Card. New components were added to the local I/O card and continuity checks were made to insure that the wiring was correct. Two of the wires from USART #2 had been crossed, connected to the wrong pins, and were corrected. Next, a power on test was done to insure that voltages were correct. The three line drivers had +11.8 volts on pin 14 and -12.1 on pin 1. The MC1488s have a flexibe range for the input power and voltages between 9 and 15 volts will work. The multibus has a +12 and -12 volt supply so this was used.

The program converted from PL/Z was used to check the I/O ports on the local card. The I/O ports were initialized and the interrupt routines accepted data from a terminal (ADM-3A) and put it in the correct location. Data destined for a port was also written on the terminal connected to that port. All this testing was done using the ICE-86A and there were never more than two terminals connected to the UNID at any time. This was both because of physical space constraints and the non availability of terminals when needed.

The converted local program was used to test the local card and as a result some testing was accomplished on it as a result. Extensive testing of this software package was not done but there are a large number of test points built in

which will make extensive testing easier. Complete testing of this package will require packets to be exchanged with other terminals connected to the four ports using both good and incorrect routing information.

Summary of UNID II Testing Procedure

This chapter presented the test procedure used in testing the hardware and software of the UNID II. Initial testing insured that the UNID II was still functioning as it was intended before modifications were attempted. The hardware testing consisted of both power off and power on tests. Software testing was conducted as each program was completed to insure it was correct and usable. The local program, converted from PL/Z to PL/M-86, proved that the local subsystem operates correctly for communications between monitors connected to two different local ports. This program tested the entire local subsystem of the UNID II.

V. Conclusions and Recommendations

The objective of this investigation was to continue the development of a Universal Network Interface Device (UNID II) based on the Intel 8086 family of microprocessors. The original UNID and the new version consist of two modules; a local module for interfacing to host computers or terminals, and a network module for interfacing to a computer network, initially this will be the AFIT DELNET. The UNID II is intended to function as a node of a computer network. The UNID II system uses an off the shelf iSBC 86/12A as the processor and wire wrap cards for both local and network I/O.

The two wire wrap cards had been started by previous investigators and circuits had been designed for the completion of the network card. The completion of the local card was straight forward and the required components were available. The network card design was fairly complete, but unfortunatly it could not be built using this design. Design changes were needed both to use the required circuits and to interface with the existing DELNET.

The UNID II had never operated in a stand alone configuration since there was no monitor program in the EPROM. All development had been done using the ICE-86A. The hardware development worked well this way and software was developed and tested for both the local and network modules. There was no way to test both modules using the ICE-86A since both used an 8086 and the ICE-86A can only interface

to one socket at a time. Installation and checkout of the monitor program on the iSBC 86/12A was an important step in development of a usable UNID II.

Conversion of PLZ programs to PL/M-86 proved to be a more difficult task than had been anticipated. There were substantial differences between the two languages which made direct conversion imposible. There were also differences in the way the serial local channels were serviced. The older UNIDs have interrupt driven receive links but the transmit side is in line code. The low level drivers were in assembly language (Z-80) for the original code and it was written in PL/M-86 for UNID II.

This effort resulted in the completion of the local module, including software that is equal to that on the original UNIDs. The network module is still not finished, but the design is now complete and it can be built. The software development should go much faster with the new design.

Recommendations

The local card is functional but there is a lot of overhead because the parallel port of the iSBC 86/12A is being used for all local accesses. A new design for local I/O, one that does not use the parallel port, is needed if the UNID II design is continued. The reason for using the parallel port was to reduce the use of the bus, but there are many other methods which could provide a better solution

to the I/O causing bus contention.

The EPROM monitor is working and there is a program to interface with the UNID II from a host computer. The interface program is not extensive and it has not been modified to work on the Intellec systems. It would be advisable to get the iSBC Intellec - iSBC 86/12A Interface and Execution Package. This package was designed to work on the Intellec with an iSBC 86/12A and it has all the bells and whistles that may be missing in the locally developed version.

Software development has been progressing very well on the Z-80 UNIDs and it is important that the UNID II not fall behind. It can be difficult to catch up and there is no reason to develop the same software at different times. The DELNET software for both the original UNIDs and UNID II should be developed concurrently since the hardware is intended to work together in the same network. The original hardware design of the UNID II missed this important point and we do not want to go our separate ways at this late stage of the game.

The UNID II is almost a reality so the temptation to go on to newer and faster processors must be avoided. This device is built with proven components and it will operate as fast as we are likely to want for some time. It is time to get the DELNET built with the original two UNIDs and the UNID II and see how the network will operate. There are bound to be problems both in hardware and software, but

let's find out where the problems are before we charge off to build bigger and faster UNIDs.

The cards that make up the UNID II should be housed in a container with at least six multibus connectors for printed circuit cards, and a power supply that can provide 15 amps on the +5 volt line and an amp for both the plus and minus 12 volt lines. The box should also have cutouts for at least five RS-232 connectors and two RS-422 connectors. Having all the cards in a single container with enough power to run them all at the same time would make development easier and it would be ready to install when the development is done. Several companies make the type of box needed and the cost should be reasonable.

Bibliography

1. 1842 EEG/EEIC An Engineering Assessment Toward Economic, Feasible and Responsive Base-Level Communications through the 1980's. Technical Report TR 78-5. Richard-Gebaur AFB, Missouri. October 1977.

2. Bertine, H.V. "Physical Level Protocols," IEEE Transactions on Communications, COM-28(4):433-444 (April 1980).

3. Borgsmiller, Michael. "The Serial Communications Interface Board," Unpublished Project Report, Wright-Patterson AFB, Ohio: School of Engineering, Air Force Institute of Technology, March 1983.

4. Cuomo, Gennaro."Continued Development of the Universal Network Interface Device," Unpublished Master's Thesis. Wright-Patterson AFB, Ohio: School of Engineering, Air Force Institute of Technology, December 1982.

5. EIA Standard RS-232C. Interface Between Data Terminal Equipment and Data communication Equipment Employing Serial Binary Data Interchange. Washington, D.C.: Electronic Industrial Association, April 1975.

6. EIA standard RS-449C. General Purpose 37-Position Interface for Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange. Washington, D.C.: Electronic Industrial Association, November 1977.

7. Fairchild Camera and Instrument Corp. "Microprocessor Products Data Book," Santa Clara, California: Fairchild Camera and Instrument Corporation, January 1983.

8. Folts, Harold C. " Revised CCITT Recommendation X.25 -1980," Technical Information Bulletin 80-5, National Communications System, Office of Technology Standards Washington D.C., August 1980. (AD A092394)

9. Geist, John W."Protocol Development for the Universal Network Interface Device." Unpublished Master's Thesis. Wright-Patterson AFB, Ohio: School of Engineering, Air Force Institute of Technology, December 1981.

10. Gravin, Andrew G."Preliminary Design of a Computer Communications Network Interface Device Using INTEL 8086 and 8089 16-Bit Microprocessors," Unpublished Master's Thesis. Wright-Patterson AFB, Ohio: School of Engineering, Air Force Institute of Technology, December 1981.

11. Hazelton, Craig H."Continued Development and Implementation of the Protocols for the Digital Engineering Laboratory Network," Unpublished Master's Thesis. Wright-Patterson AFB, Ohio: School of Engineering, Air Force Institute of Technology, March 1981.

12. Intel Corporation."Component Data Catalog," Santa Clara, California: Intel Corporation, January 1981.

13. Intel Corporation."iSBC 86/12A Single Board Computer Hardware Reference Manual," Manual Order No. 9800645.

14. Intel Corporation."iAPX 86/88, 186/188 User's Manual Programmer's Reference,"Santa Clara, California: Intel Corporation, May 1983.

15. Intel Corporation."ICE-86A/ICE-88A Microsystems In-Circuit Emulator Operating Instructions for ISIS-II Users," Manual Order No. 162554-002, 1982.

16. Intel Corporation. " ISIS-II User's Guide," Manual Order No. 9800306-05, 1979.

17. Intel Corporation."ISIS-II PL/M Compiler Operator's Manual," Manual Order No. 9800478A.

18. Intel Corporation. " PL/M-86 Programming Manual," Manual Order No. 9800466A.

19. Intel Corporation. " MCS-86 Software Development Utilities Operating Instructions for ISIS-II Users," Manual Order No. 9800639B.

20. Jigour, Robin. " Prototyping with the 8089 I/O Processor," Intel Application Note AP-89, May 1980.

21. Palmer, Donald E." Design of a Prototype Universal Network Interface Device Using INTEL 8086 and 8089 16-Bit Microprocessors,"Unpublished Master's Thesis. Wright-Patterson AFB, Ohio: School of Engineering, Air Force Institute of Technology, December 1982.

22. Papp, Charles E."Prototype DELNET Using the Universal Network Interface Device,"Unpublished Master's Thesis. Wright-Patterson AFB, Ohio: School of Engineering, Air Force Institute of Technology,

December 1981.

23. Phister, Paul W Jr. "Protocol Standard and Implementation Within the Digital Engineering Laboratory Computer Network (DELNET) Using the Universal Network Interface Device (UNID)," Unpublished Master's Thesis. Wright-Patterson AFB, Ohio: School of engineering, Air Force Inststute of Technology, December 1983.

24. Sluzevich, Sam C." Preliminary Design of a Universal Network Interface Device," Unpublished Master's Thesis. Wright-Patterson AFB, Ohio: School of Engineering, Air Force Institute of Technology, December 1978.

25. Tanenbaum, Andrew S."Network Protocols," Computing Surveys 13(4):453-489 (December 1981).

26. Tanenbaum, Andrew S. Computer Networks. Englewood Cliffs, New Jersey: Prentice-Hall Inc.,1981.

27. Weinberg, Victor. Structured Analysis. New York: Yourdon Press, 1979.

28. Witt, Michael."An Introduction to Layered Protocols," Byte:385-398, September 1983.

29. Zilog, Inc. PLZ User Guide (03-3096-01), Manufacturer's data. Cupertino, California: Zilog, Inc.,July 1979.

30. Zimmermann, Hubert." OSI Reference Model - The ISO Model of Architecture Open Systems Interconnections," IEEE Transactions on Communications, COM-28(4): 425-432, (April 1980).

Appendix A

UNID II Data Flow Diagrams (Ref 10:26-34)

This appendix contains the Data Flow Diagrams (DFD) for the UNID II which were developed in a previous thesis effort. These DFDs show the UNID II message processing functions and the internal flow of messages between the local and network I/O ports. The Data Flow Diagrams are presented as follows:

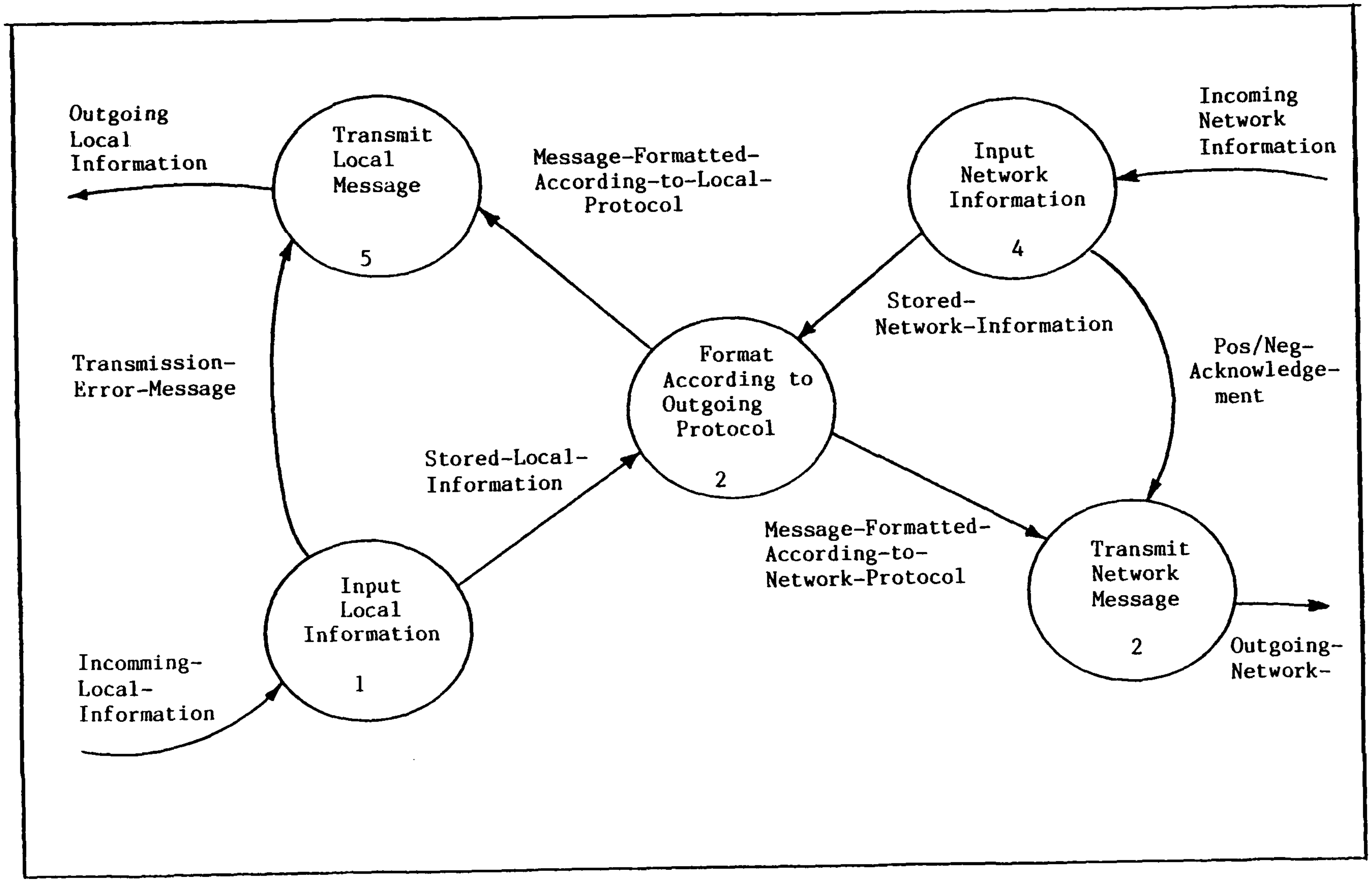


Figure A-1. UNID II Overview (Ref 10)

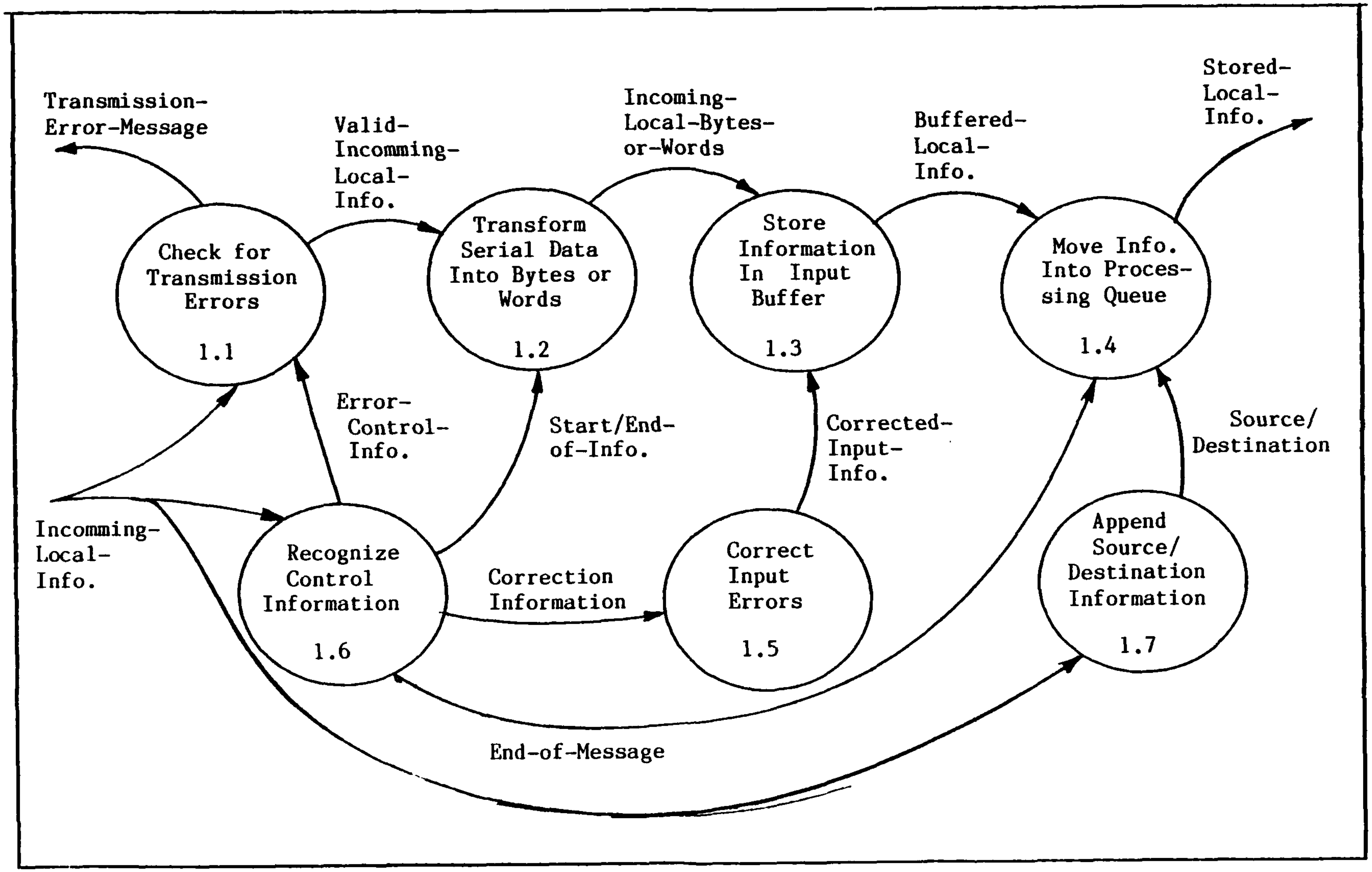


Figure A-2. Input Local Information (Ref 10)

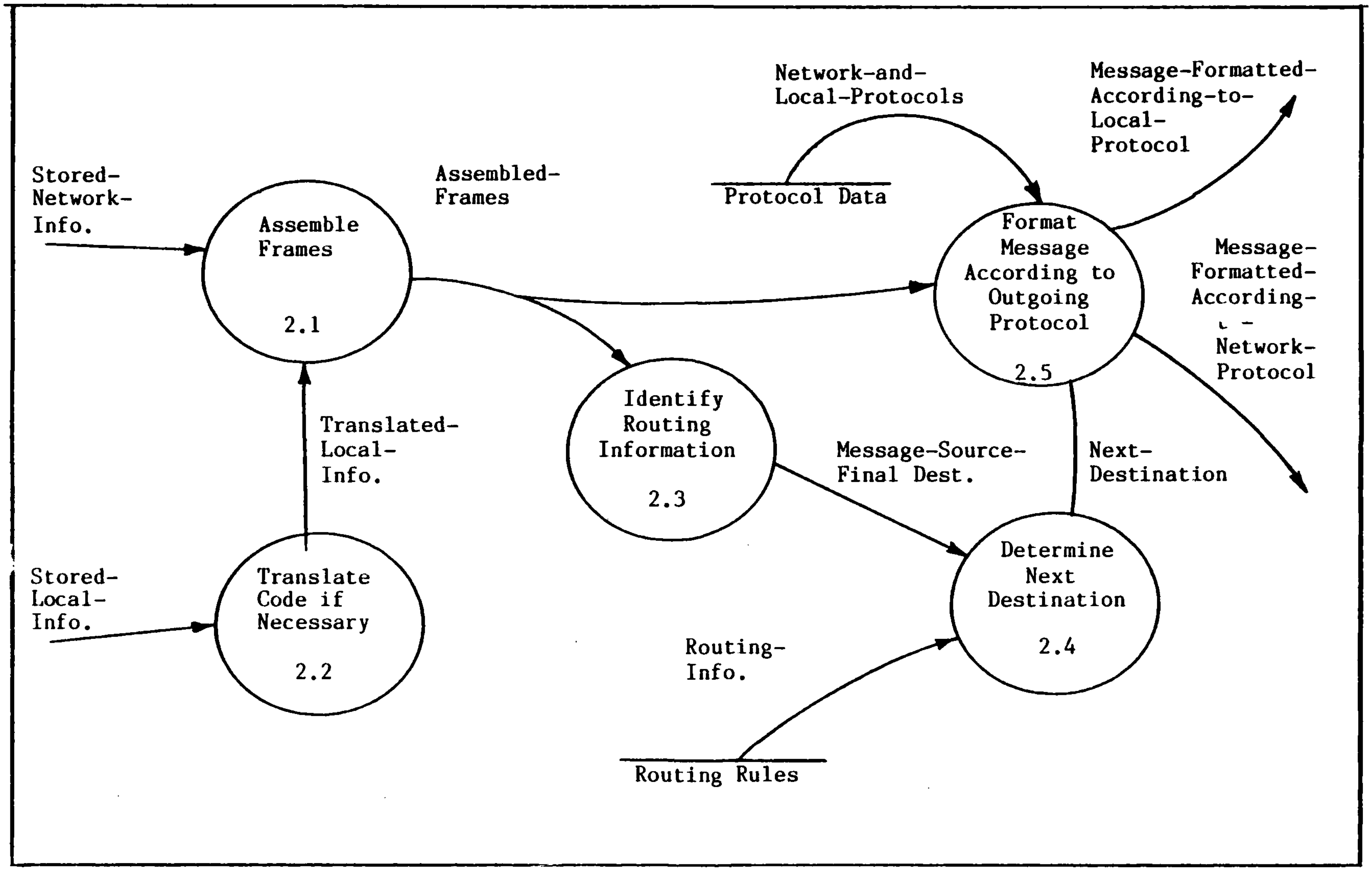


Figure A-3. Format According to Outgoing Protocol (Ref 10)

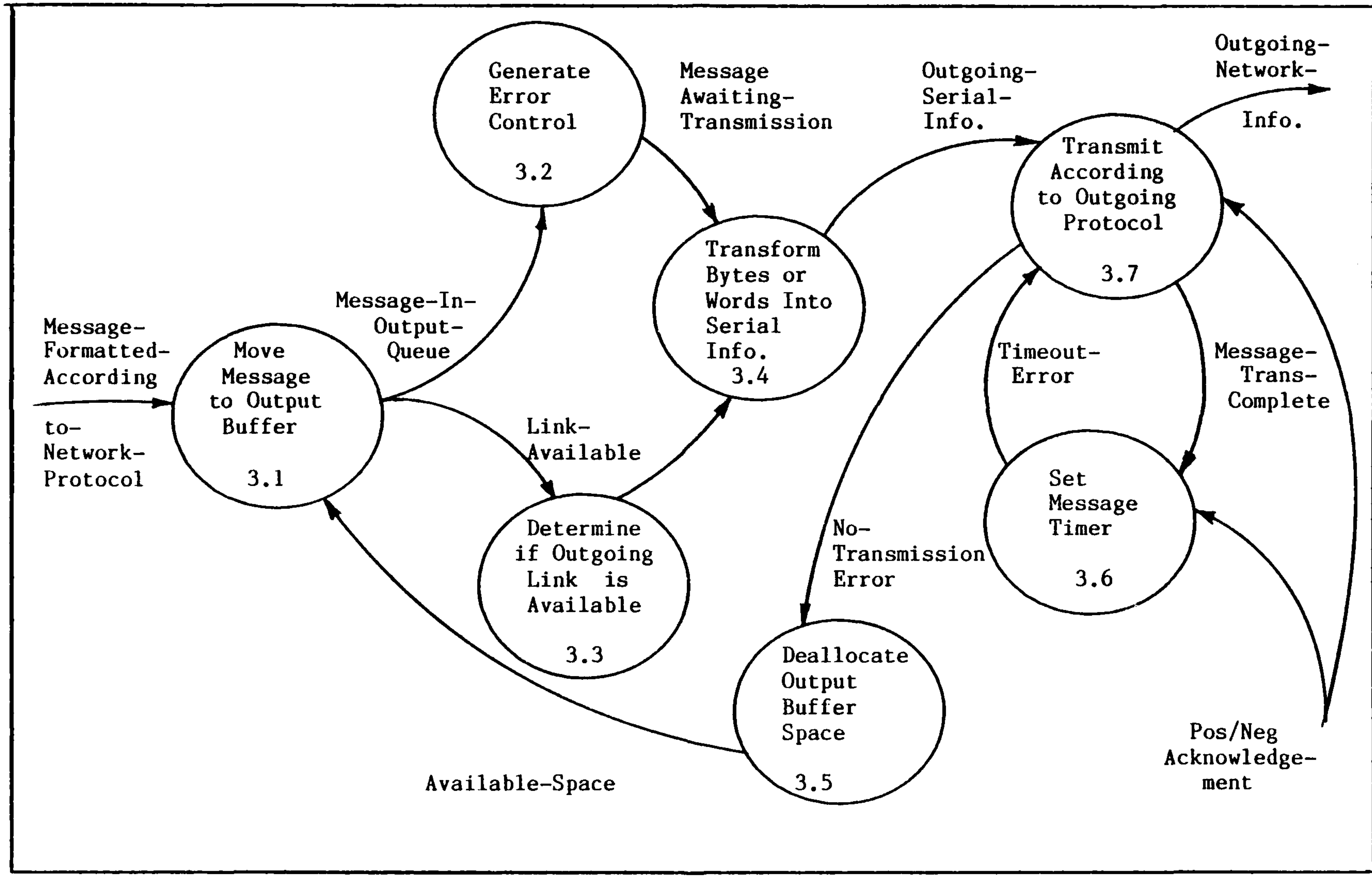


Figure A-4. Transmit Network Message (Ref 10)

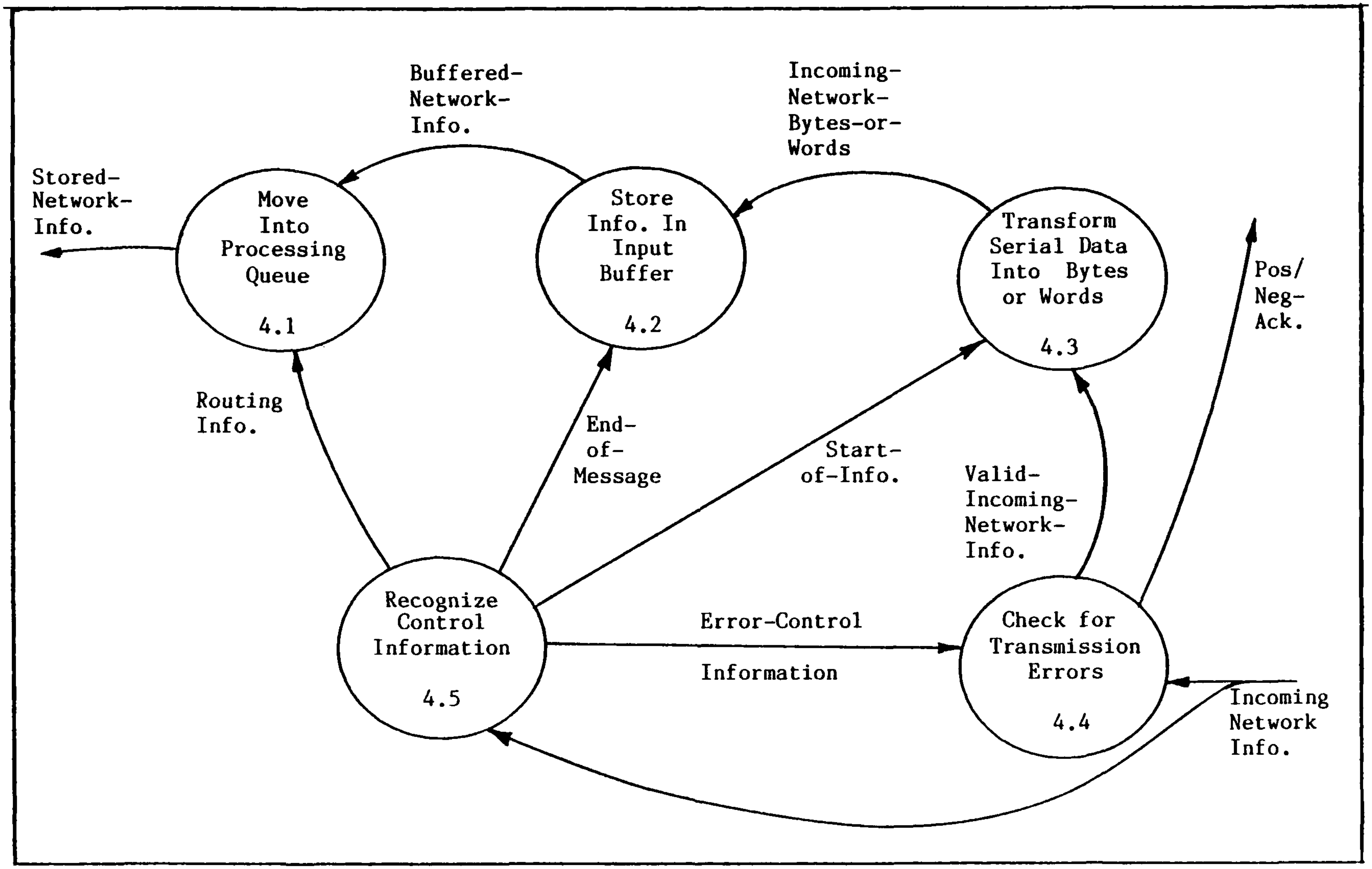


Figure A-5. Input Network Information (Ref 10)

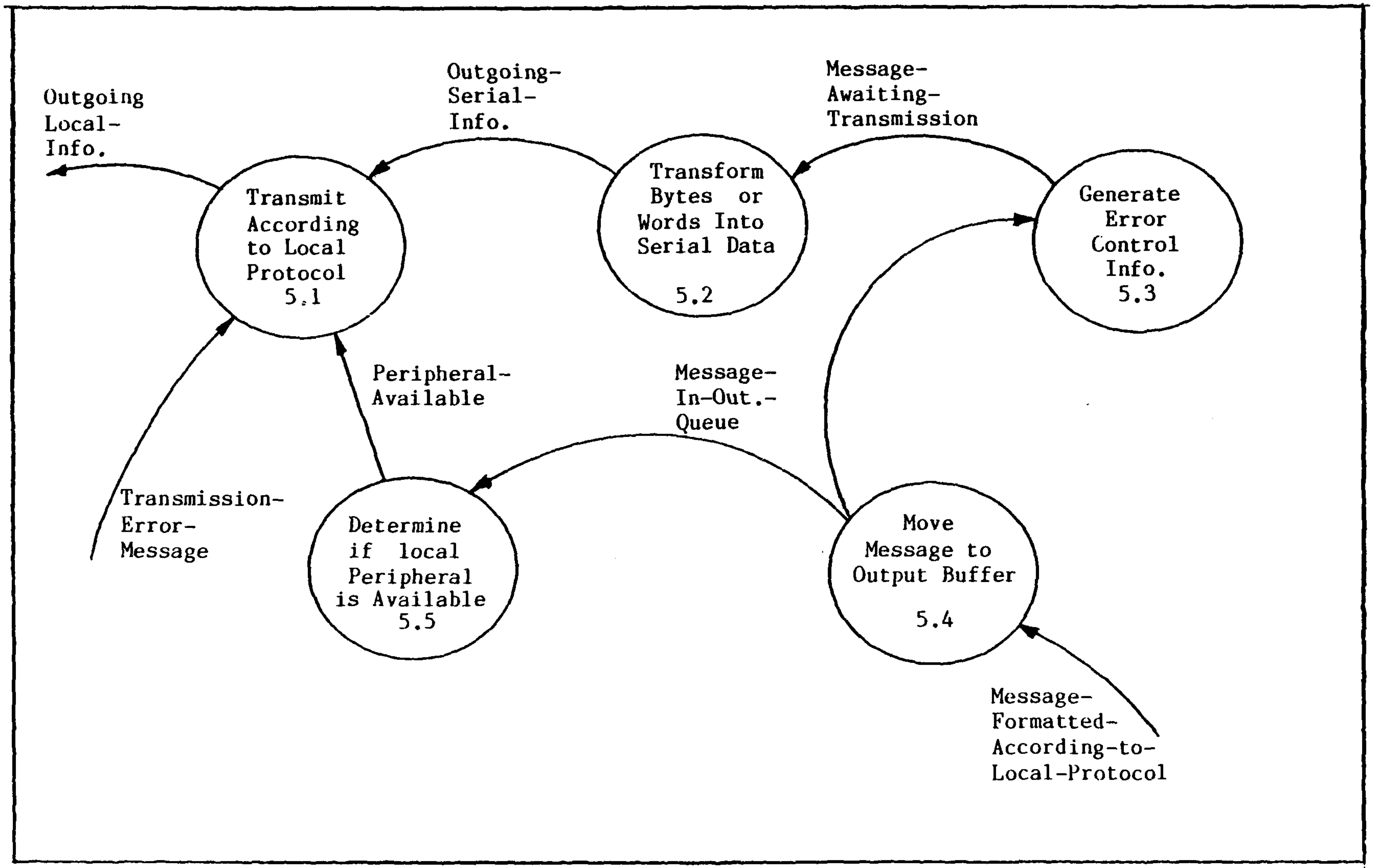


Figure A-6. Transmit Local Message (Ref 10)

Appendix B

RS-232 and RS-422 Signals

The physical layer will use the RS-232 standard on the local side and the RS-422 standard on the network links. Each standard has a large number of signals specified and not all are used by the UNIDs. The following tables indicate the signals that have been selected for implementation. Direction is not shown since the UNID II has jumpers to select either the DCE or DTE configuration.

The local side, interfacing with host computers and/or terminals, RS-232 will be used. Table B-1 indicates which pins from the RS-232 standard have been implemented in the DELNET.

Table B-1. RS-232 Pin Assignments

RS-232 Pin Number	Function	Implemented in DELNET
1	Protective Ground	X
2	Transmitted Data	X
3	Received Data	X
4	Request to Send	X
5	Clear to Send	X
6	Data Set Ready	X
7	Signal Ground	X
8	Received Line Signal Detector	X
9	Reserved for Testing	
10	Reserved for Testing	
11	Unassigned	
12	Secondary Receive Signal Detect	
13	Secondary Clear to Send	
14	Secondary Transmit Data	
15	Transmit Signal Element Timing	
16	Secondary Receive Data	
17	Receive Signal Element Timing	
18	Unassigned	
19	Secondary Request to Send	
20	Data Terminal Ready	X
21	Signal Quality Detector	
22	Ring Detector	
23	Data Signal Rate Selector(DTE)	
24	Data Signal Rate Selector(DCE)	
25	Unassigned	

On the network side the UNIDs are connected in a double ring configuration using the RS-422 standard. Table B-2 gives the pin assignments and indicates which are currently implemented in the DELNET. As with the RS-232, direction is not indicated since they may be jumpered to be either a DCE

or DTE.

Table B-2. RS-422 Pin Assignments

Pin Number	Function	Implemented
1	Shield	X
2	Signal Rate Indicator	
3	Spare	
4	Send Data	X
5	Send Timing	X
6	Received Data	X
7	Request to Send	X
8	Receive Timing	X
9	Clear to Send	X
10	Local Loopback	
11	Data Mode	
12	Terminal Ready	X
13	Receiver Ready	X
14	Remote Loopback	
15	Incoming Call	
16	Select Frequency Signaling Rate Indicator	
17	Terminal Timing	
18	Test Mode	
19	Signal Ground	X
20	Receive Common	
21	Spare	
22	Send Data	X
23	Send Timing	X
24	Receive Data	X
25	Request to Send	X
26	Receive Timing	X
27	Clear to Send	X
28	Terminal in Service	
29	Data Mode	
30	Terminal Ready	X
31	Receiver Ready	X
32	Select Standby	
33	Signal Quality	
34	New Signal	
35	Terminal Timing	
36	Standby Indicator	
37	Send Common	

Appendix C

Monitor Program Listings

This section contains the programs to convert from the Intel extended HEX format to the standard HEX format and the interface program for the monitor. The monitor program is not contained here because it is copyrighted by Intel. The monitor program listing can be obtained from Intel as part of a package to interface the iSBC 86/12A card with the Intellec Development System.

Table of Contents

```c
/***********************************************************
*                                                          *
*       DATE: 24 September 1983                            *
*       VERSION: 1.4                                       *
*       NAME: HEXCONV                                      *
*       FUNCTION: Takes a file in the extended hex format  *
*               and converts it to the standard Intel hex  *
*               format. Creates a file in the standard hex *
*               format.                                    *
*       SYSTEM INFORMATION: Written in Computer Innovations *
*               C-86 for use on CPM-86.                    *
*       MODULES USED: CHECKHEX, HEXVAL, INTASCI, OFFSET,   *
*               CALCADDR, REDUCE, HEXASCI, GETBITS.        *
*       AUTHOR: William F. Matheson                        *
*       HISTORY: Original                                  *
***********************************************************/

#include        "stdio.h"

        main()

        {

            FILE *inhex, *outhex;

            char  code[50], adrloc[5];

            int  checksum, length, stop, actadd, newsum, i, j;

            long  offval, exaddr;

            inhex = fopen("EPROM.HEX","r");

            outhex = fopen("MON.HEX","w");

            stop = 1; /* used to loop until file complete */

            offval = 0; /* set extended address to zero */

            length = 0; /* set length of line to zero */

            checksum = 0; /* begin with zero in checksum */

            while (stop) /* loop until input file is empty */

            {

                fgets (code, 50, inhex); /* get a line of code */

                length = strlen(code); /* how long is this line */

                if(length >= 13) /* only end of file line is shorter */

                {
```

```
            if(code[8] == '2') /* indicates line contains an
                                  extended address field */
            {

                offval = offset(code,5); /* calculate the
                                            address in hex */
            }

            else

            {

               exaddr = calcaddr(code,offval, 4); /* convert
                                                     to hex addr */
               actadd = reduce(exaddr); /* put into a 64K
                                           address range */
               intasci(actadd, adrloc); /* convert back to
                                           ascii for output */
               for(i=0; i <= 3; i++)

               {

                  code[i+3] = adrloc[i]; /* put the new address
                                            in the output str */
               }

               checksum = checkhex(code, length); /* calculate
                                                     the new checksum */
               for(j = 1; j <= 2; j++)

               {

                  newsum = getbits(checksum,(11 - (j*4)),4);

                  code[length - (4 - j)] = hexasci(newsum);
                      /* converts them to ascii */
               }

          fputs(code,outhex); /* write the new line to the
                                 output file */
            }

        }

        else

            stop = 0; /* line was less than 13 */

    }

}
```

```
/************************************************************
*                                                           *
*     DATE: 15 May 1983                                     *
*     VERSION: 1.0                                          *
*     NAME: checkhex                                        *
*     MODULE NUMBER: 1                                      *
*     FUNCTION:  Converts the two character ascii represet- *
*                ation of the memory address to a hex or    *
*                binary value for use in the calculations   *
*                needed in the program.                     *
*                                                           *
*     INPUTS:  code and length                              *
*     OUTPUTS:  comp + 1.                                   *
*     GLOBAL VARIABLES USED: none                           *
*     GLOBAL VARIABLES CHANGED: none                        *
*     GLOBAL TABLES USED:  none                             *
*     GLOBAL TABLES CHANGED: none                           *
*     FILES READ: none                                      *
*     FILES WRITTEN: none                                   *
*     MODULES CALLED:  hexval                               *
*     CALLING MODULES: main                                 *
*     AUTHOR:    William F. Matheson                        *
*     HISTORY:   Original                                   *
*                                                           *
************************************************************/
checkhex(code, length)

char *code;

int length;

{

    int i, number, sum, comp, hival, loval;

    sum = 0;

    for(i = 1; i < length - 3; i++)

    {

        hival = hexval(code[i]); /* convert high byte */
        hival = hival * 16; /* moves it up one digit */
        i++;
        loval = hexval(code[i]); /* convert low byte */
        number = hival + loval; /* add the two */
        sum = sum + number; /* keep running total */

    }

    comp = sum ^ 017777; /* complement the sum */

return(comp + 1); /* add one to produce twos complement */
}
```

```
/*************************************************************
*                                                            *
*       DATE: 15 May 1983                                    *
*       VERSION: 1.0                                         *
*       NAME: hexval                                         *
*       MODULE NUMBER: 2                                     *
*       FUNCTION:  Converts an ascii character to hex.       *
*                                                            *
*                                                            *
*                                                            *
*                                                            *
*       INPUTS: digit                                        *
*       OUTPUTS: newval                                      *
*       GLOBAL VARIABLES USED: none                          *
*       GLOBAL VARIABLES CHANGED: none                       *
*       GLOBAL TABLES USED: none                             *
*       GLOBAL TABLES CHANGED: none                          *
*       FILES READ: none                                     *
*       FILES WRITTEN: none                                  *
*       MODULES CALLED:  none                                *
*       CALLING MODULES: checkhex                            *
*       AUTHOR:    William F. Matheson                       *
*       HISTORY:   Original                                  *
*                                                            *
*************************************************************/
hexval(digit)

int digit;

{

    int check, newval;

    newval = digit & 0017; /* picks out low half of the
                              number */

    check = digit & 000100; /* determine if the ascii code is a
                               number or a letter */

    if(check > 0)

        return(newval + 9); /* if a letter add 9 */

    else

        return(newval);
}
```

```
/***************************************************************
*                                                              *
*       DATE: 3 Oct 1983                                       *
*       VERSION: 1.0                                           *
*       NAME: intasci                                          *
*       MODULE NUMBER: 3                                       *
*       FUNCTION:  Converts the integer value for the actual   *
*                  address to four ascii characters which      *
*                  are to be placed in the output file and     *
*                  also used for the checksum calculation.     *
*                                                              *
*       INPUTS:  actadd and adrloc                             *
*       OUTPUTS:  none                                         *
*       GLOBAL VARIABLES USED: none                            *
*       GLOBAL VARIABLES CHANGED: none                         *
*       GLOBAL TABLES USED:  none                              *
*       GLOBAL TABLES CHANGED: none                            *
*       FILES READ: none                                       *
*       FILES WRITTEN: none                                    *
*       MODULES CALLED:  hexasci                               *
*       CALLING MODULES: main                                  *
*       AUTHOR:    William F. Matheson                         *
*       HISTORY:   Original                                    *
*                                                              *
***************************************************************/
intasci(actadd, adrloc)

int  actadd;

char  *adrloc;

{

   int i;

   char temp;

   for(i = 0; i <= 3; i++)

   {

      temp = getbits(actadd,(15 - (i*4)), 4);

      adrloc[i] = hexasci(temp); /* converts the integer to
                                    ascii for output */

   }

}
```

```
/****************************************************************
*                                                               *
*       DATE: 2 October 1983                                    *
*       VERSION: 1.0                                            *
*       NAME:  offset                                           *
*       MODULE NUMBER: 4                                        *
*       FUNCTION:   Takes the ascii values and produces a       *
*                  long integer that is the value of the        *
*                  base which is used for address calculations.*
*                                                               *
*                                                               *
*       INPUTS: code and lenstr.                                *
*       OUTPUTS: offval.                                        *
*       GLOBAL VARIABLES USED: none                             *
*       GLOBAL VARIABLES CHANGED: none                          *
*       GLOBAL TABLES USED: none                                *
*       GLOBAL TABLES CHANGED: none                             *
*       FILES READ: none                                        *
*       FILES WRITTEN: none                                     *
*       MODULES CALLED: hexval                                  *
*       CALLING MODULES: main                                   *
*       AUTHOR:    William F. Matheson                          *
*       HISTORY:   Original                                     *
*                                                               *
****************************************************************/
offset(code, lenstr)

char  *code;

int  lenstr;

{

   int  power, count, value;

   long  offval;

   offval = 0;

   for(count = 1; count <= 4; count++)

   {

      value = hexval(code[count + 8]); /* this is to convert the
                                          ascii char to an int */
      for(power = lenstr - count; power > 0; power--)

         value = value * 16; /* this puts the portion of the int
                                in the correct position */
      offval = offval + value; /* sum of the number in the addr
                                  field converted to an int */
   }
   return(offval);
}
```

```
/****************************************************************
*                                                               *
*       DATE: 2 October 1983                                    *
*       VERSION: 1.0                                            *
*       NAME: calcaddr                                          *
*       MODULE NUMBER: 5                                        *
*       FUNCTION:   Determines the actual address for the       *
*             code using both the base address and the offset. *
*                                                               *
*                                                               *
*                                                               *
*       INPUTS: code, offval, and lenstr.                       *
*       OUTPUTS: exaddr                                         *
*       GLOBAL VARIABLES USED: none                             *
*       GLOBAL VARIABLES CHANGED: none                          *
*       GLOBAL TABLES USED: none                                *
*       GLOBAL TABLES CHANGED: none                             *
*       FILES READ: none                                        *
*       FILES WRITTEN:  none                                    *
*       MODULES CALLED:  hexval                                 *
*       CALLING MODULES: main                                   *
*       AUTHOR:    William F. Matheson                          *
*       HISTORY:   Original                                     *
*                                                               *
****************************************************************/
calcaddr(code,offval,lenstr)
char  *code;
long  offval;
int  lenstr;
{

   int  power,count,value;

   long  exaddr;

   exaddr = offval;

   for(count = 1;count <= 4; count++)

   {

     value = hexval(code[count + 2]); /* convert ascii to int */

     for(power = lenstr - count; power >0; power--)

       value = value * 16; /* scales the magnitude of each char
                              when calculating the int number */
     exaddr = exaddr + value;

   }

   return(exaddr);

}
```

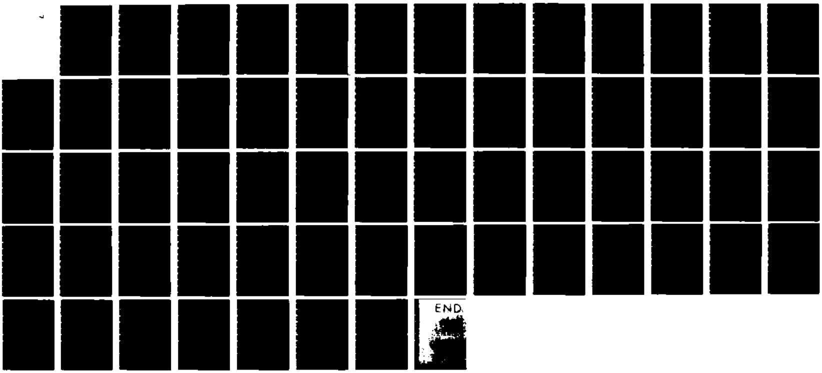
```
/**************************************************************
*                                                             *
*      DATE: 2 October 1983                                   *
*      VERSION: 1.0                                           *
*      NAME: reduce                                           *
*      MODULE NUMBER: 6                                       *
*      FUNCTION:  To remove the high order bits so the        *
*               address will fit into a 64K address space.   *
*               The normal Intel hex file uses only 64K.      *
*                                                             *
*                                                             *
*      INPUTS: exaddr                                         *
*      OUTPUTS: actadd                                        *
*      GLOBAL VARIABLES USED: none                            *
*      GLOBAL VARIABLES CHANGED: none                         *
*      GLOBAL TABLES USED: none                               *
*      GLOBAL TABLES CHANGED: none                            *
*      FILES READ: none                                       *
*      FILES WRITTEN: none                                    *
*      MODULES CALLED: none                                   *
*      CALLING MODULES: main                                  *
*      AUTHOR:    William F. Matheson                         *
*      HISTORY:   Original                                    *
*                                                             *
**************************************************************/
reduce(exaddr)

long  exaddr;

{

   int  temp,actadd;

   temp = 0;

   actadd = 0;

   if(exa dr > 65536) /* see if address is already less than
                         64K. If not make it less */
   {

      temp = exaddr / 65536; /* divide with int result */

      actadd = (exaddr - (temp * 65536)); /* subtract any integer
                                 multiple of 64K to give a result
                                 that is less than 64K */
   }

   else

      actadd = exaddr;

   return(actadd);
}
```

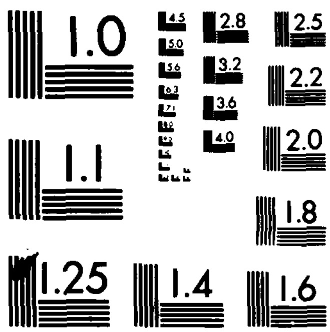
```
/*************************************************************
*                                                            *
*       DATE: 2 October 1983                                 *
*       VERSION: 1.0                                         *
*       NAME: hexasci                                        *
*       MODULE NUMBER: 8                                     *
*       FUNCTION:  A hex value is passed into this module    *
*               and the ascii character representation is    *
*               returned.                                    *
*                                                            *
*                                                            *
*       INPUTS: hexnum                                       *
*       OUTPUTS: ascinum                                     *
*       GLOBAL VARIABLES USED:none                           *
*       GLOBAL VARIABLES CHANGED: none                       *
*       GLOBAL TABLES USED: none                             *
*       GLOBAL TABLES CHANGED: none                          *
*       FILES READ: none                                     *
*       FILES WRITTEN: none                                  *
*       MODULES CALLED: none                                 *
*       CALLING MODULES: main                                *
*       AUTHOR:    William F. Matheson                       *
*       HISTORY:   Original                                  *
*                                                            *
*************************************************************/
hexasci(hexnum)

char hexnum;

{

   char  ascinum:

   if(hexnum <  9) /* does the number get represented as a number
                      if so convert by adding 48 */

       ascinum = hexnum + 48;

   else

      ascinum = hexnum + 55; /* if it will be a letter the add
                                55 instead */

   return(ascinum);

}
```

```
/****************************************************************
*                                                               *
*       DATE: 2 October 1983                                    *
*       VERSION: 1.0                                            *
*       NAME:   getbits                                         *
*       MODULE NUMBER: 9                                        *
*       FUNCTION:  This module takes a binary value and         *
*                 returns the bits for the portion asked        *
*                 for in the calling parameters.                *
*                                                               *
*                                                               *
*       INPUTS: actadd, start, and count.                       *
*       OUTPUTS: an unsigned bit field.                         *
*       GLOBAL VARIABLES USED: none                             *
*       GLOBAL VARIABLES CHANGED: none                          *
*       GLOBAL TABLES USED: none                                *
*       GLOBAL TABLES CHANGED: none                             *
*       FILES READ: none                                        *
*       FILES WRITTEN:  none                                    *
*       MODULES CALLED: none                                    *
*       CALLING MODULES: main                                   *
*       AUTHOR:    William F. Matheson                          *
*       HISTORY:   Copied from page 45 of "The C Programming    *
*               Language" by Kernighan and Ritchie.             *
****************************************************************/
getbits(actadd,start,count)

unsigned  actadd,start,count;

{

   return((actadd >> (start + 1 - count)) &~ (~0 << count));
      /* masks out the bits asked for by the calling routine
         and returns them */

}
```

AD-A138 118 CONTINUED DEVELOPMENT OF A UNIVERSAL NETWORK INTERFACE DEVICE USING THE I..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. W F MATHESON 2/2

UNCLASSIFIED 15 DEC 83 AFIT/GE/EE/83D-42 F/G 9/2 NL




1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

```
/***********************************************************************
*      DATE: 17 OCTOBER  1983                                          *
*      VERSION: 1.2                                                    *
*      NAME: INTER                                                     *
*      FUNCTION: This program is meant to interface with the Intel     *
*                SBC 86/12A monitor program. It will act as a dumb     *
*                terminal in all cases except when a hex file is to    *
*                downloaded for execution in the iSBC 86/12A. In this  *
*                case it must strip off the file information and open  *
*                the file.                                             *
*      SYSTEM INFORMATION: The program is written in the Computer      *
*                Innovations C-86 compiler for CPM-86 for a CompuPro   *
*                8085/8088 Dual Processor card and the interfacer 4.   *
*      AUTHOR: William F. Matheson                                     *
*      HISTORY: Original                                               *
***********************************************************************/

#include  "stdio.h"

#define   FALSE   0

#define   TRUE   377

#define   EOF   '032'

#define   PORT   020      /* base address of the interfacer 4 */

#define   DTRON  027      /* turn on the dtr */

#define   DTROFF 025      /* turn the dtr off */

#define   MODEONE  356   /* mode word one - 1 stop bit, no parity, 8 info
                            bits, async x 16 clock rate */
#define   MODETWO  176   /* mode word two - internal tx & rx clock, baud
                            rate 9600 baud  */
#define   SETCHAR '\125'/* character U to set baud rate  */

   main()

   {

      char  status, nextchar, typechar, nochar, okey;

      initport(PORT); /* set initial parameters for USART  */

      outportb(PORT + 7,06);  /* insure correct port is selected */

      nochar = inportb(PORT); /* clear USART buffer */

      outportb(PORT, SETCHAR); /* send char to establish baud rate */

      outportb(PORT + 3,DTRON); /* allow the 86/12A to transmit */

      msdelay(1); /* delay for about 1 msec */
```

```
outportb(PORT + 3,DTROFF); /* inhibit the 86/12A transmit */

status = inportb(PORT + 1); /* check for USART status */

while((status & 02) != 02)

{

    status = inportb(PORT + 1); /* loop to wait for receive
                                   character */
}

dumbterm(); /* When a character is received go to dumb term */

for(;;)  /* after initial setup stay in this loop forever */

{

   nextchar = getchar(); /* get input command */

   if(nextchar == 'L') /* check to see if it is the LOAD command
                          or any other command */
   {

      loader(nextchar); /* It was a LOAD command so look for
                           file information */
      nextchar = '\n';

   }

   else

   {

      while(nextchar != '\n')  /* it was not LOAD so get rest of
                                  the command string */
      {

         outportb(PORT,nextchar); /* send each char as it is input
                                     to the terminal */
         typechar = rxchar();  /* see if the response from the
                                  86/12A is an error code */
         if(typechar)

            nextchar = getchar(); /* no error so continue */

         else

         {

            nextchar = '\n';

            outportb(PORT,nextchar); /* there was an error so
```

```
                                    stop processing input */
            }
          }
        }
        nextchar = getchar(); /* get the next char from terminal */
      }
    }
```

```
/***********************************************************************
*      DATE: 23 OCTOBER 1983                                           *
*      VERSION: 1.1                                                    *
*      NAME: INITPORT                                                  *
*      MODULE NUMBER: 1                                                *
*      FUNCTION: Set up the USART for communications with the iSBC     *
*                86/12A monitor routine.                               *
*      INPUTS: PORT BASE ADDRESS                                       *
*      OUTPUTS: NONE                                                   *
*      GLOBAL VARIABLES USED: NONE                                     *
*      GLOBAL VARIABLES CHANGED: NONE                                  *
*      FILES READ: NONE                                                *
*      FILES WRITTEN: NONE                                             *
*      MODULES CALLED: NONE                                            *
*      CALLING MODULES: MAIN                                           *
*      AUTHOR: William F. Matheson                                     *
*      HISTORY: ORIGINAL                                               *
***********************************************************************/
  initport(port)

  char  port;

  {

     outportb(port + 7, 04); /* select serial channel to use */

     msdelay(200); /* DELAY FOR SETUP */

     outportb(port + 2, MODEONE); /* mode word one - 1 stop bit, no */
                                  /* parity, 8 info bits, async x 16 */
     msdelay(200);

     outportb(port + 2, MODETWO); /* internal tx and rx clock, 9600baud */

     msdelay(200);

     outportb(port + 3, DTROFF); /* rts high dtr low, rx and tx enable */

     msdelay(200);

     return;

  }
```

```
/***************************************************************************
*      DATE: 19 OCTOBER 1983                                               *
*      VERSION: 1.0                                                        *
*      NAME: MSDELAY                                                       *
*      MODULE NUMBER: 2                                                    *
*      FUNCTION: Provides a delay of about lmsec times the input value     *
*      INPUTS: N, AN INTEGER                                               *
*      OUTPUTS: NONE                                                       *
*      GLOBAL VARIABLES USED: NONE                                         *
*      GLOBAL VARIABLES CHANGED: NONE                                      *
*      FILES READ: NONE                                                    *
*      FILES WRITTEN:NONE                                                  *
*      MODULES CALLED: NONE                                                *
*      CALLING MODULES: MAIN, INITPORT, LOADER                             *
*      AUTHOR: William F. Matheson                                         *
*      HISTORY: ORIGINAL                                                   *
***************************************************************************/
   msdelay(n)

   char  n;

   {

      char  i,j;

      for(i=0;i<=n;i++)

      {

         for(j=0;j<=55;j++) /* loop for delay */

         ;

      }

      return;

   }
```

```
/***************************************************************************
*     DATE: 19 OCTOBER 1983                                                *
*     VERSION: 1.0                                                         *
*     NAME: GETLINE                                                        *
*     MODULE NUMBER: 3                                                     *
*     FUNCTION: Used to read the command line for a LOAD instruction       *
*     INPUTS: S, CHARACTER STRING POINTER: LIM, MAXIMUM LINE LENGTH        *
*     OUTPUTS: NONE                                                        *
*     GLOBAL VARIABLES USED: NONE                                          *
*     GLOBAL VARIABLES CHANGED: NONE                                       *
*     FILES READ: NONE                                                     *
*     FILES WRITTEN: NONE                                                  *
*     MODULES CALLED: NONE                                                 *
*     CALLING MODULES: LOADER                                              *
*     AUTHOR: William F. Matheson                                          *
*     HISTORY : Modified version of routine in The C Programming           *
*               Language by Kernighan and Ritchie                          *
***************************************************************************/
  getline(s,lim)

  char *s;

  int  lim;

  {

     int  c,i;

     for(i=1;i < lim-1 &&(c=getchar()) != EOF && c != '\n';++i)

        s[i] = c; /* continue to get characters and place them in the
                     string until an EOF, newline, or limit is
                     reached  */

     if(c == '\n')

     {

        s[i] = c;

        ++i;

     }

     s[i] = '\0';

     return;

  }
```

```
/***********************************************************************
*       DATE: 20 OCTOBER 1983                                          *
*       VERSION: 1.4                                                   *
*       NAME: LOADER                                                   *
*       MODULE NUMBER: 4                                               *
*       FUNCTION: This module determines the file to be loaded, opens *
*                 it and sends it to the iSBC 86/12A.                  *
*       INPUTS: FIRST, A SINGLE CHARACTER                              *
*       OUTPUTS: NONE                                                  *
*       GLOBAL VARIABLES USED: NONE                                    *
*       GLOBAL VARIABLES CHANGED : NONE                                *
*       FILES READ: A HEX FILE DESIGNATED BY THE USER FROM THE KEYBOARD *
*       FILES WRITTEN: NONE                                            *
*       MODULES CALLED: GETLINE, MSDELAY                               *
*       CALLING MODULE: MAIN                                           *
*       AUTHOR: William F. Matheson                                    *
*       HISTORY: ORIGINAL                                              *
***********************************************************************/

   loader(first)

   char  first;

   {

      char  i,t,code[50],stat,filename[14],whichfle[28],more,good,k;

      int  length;

      FILE  *hexfile; /* will be used for the output file */

      whichfle[0] = first; /* file name is input */

      getline(whichfle,30);

      for(i=0;i<6 && (whichfle[i]) != ',';i++)

         t = i;

      if(isalpha(whichfle[t+1])) /* insure that filename is alpha */

      {

         filename[0] = '"';

         for(i=0;i< 16 && whichfle[i] != '\n';++i)

            filename[i] = whichfle[i + t]; /* put file name in correct
                                              format to open the file */

         filename[i + 1] = '"';

         filename[i + 2] = '\n';
```

```
hexfile = fopen(filename,"r"); /* open the designated file */

outportb(PORT,'L'); /* let the 86/12A know we are sending
                       a hex file on the serial port */

good = rxchar(); /* check for error code from 86/12A */

if(!good)

   goto error; /* if error code stop processing */

outportb(PORT, 'S'); /* indicates serial port will be used */

good = rxchar();

if(!good)

   goto error; /* stop if error code received */

outportb(PORT, '\n'); /* ready to start */

good = rxchar();

if(!good)

   goto error; /* another error check */

 more = TRUE;

i = 1;

while(!EOF)

{

   fgets(code,50,hexfile);

   length = strlen(code); /* send file until it is empty */

   while(more)

   {

      stat = inportb(PORT + 1);

      if((stat & 200) == 200) /* check for DSR */

      {

         outportb(PORT,code[i]); /* send one char at a time */

         msdelay(40); /* delay about 40 msec */

      outportb(PORT + 3,DTRON); /* let 86/12A send error code
```

```
                                        if needed */
                for(k = 0;k <= 28; k++)
                    ;        /* delay 1/2 millisec */
                stat = inportb(PORT + 1);
                if((stat & 02) == 02) /* check for received char */
                {
                   printf("\n-"); /* error code received so stop
                                      sending the file */
                   code[i + 1] = EOF;
                }
                outportb(PORT + 3, DTROFF);  /* turn dtr off */
                i = i + 1;
                if(i == length) /* check to see if this line is done */
                   more = FALSE;
             }
             else
                stat = inportb(PORT + 1);
             }
          }
       }
       else
       {
          printf("the file name is incorrect start over\n");
          return(TRUE);
       }
error:  return(FALSE);
    }
```

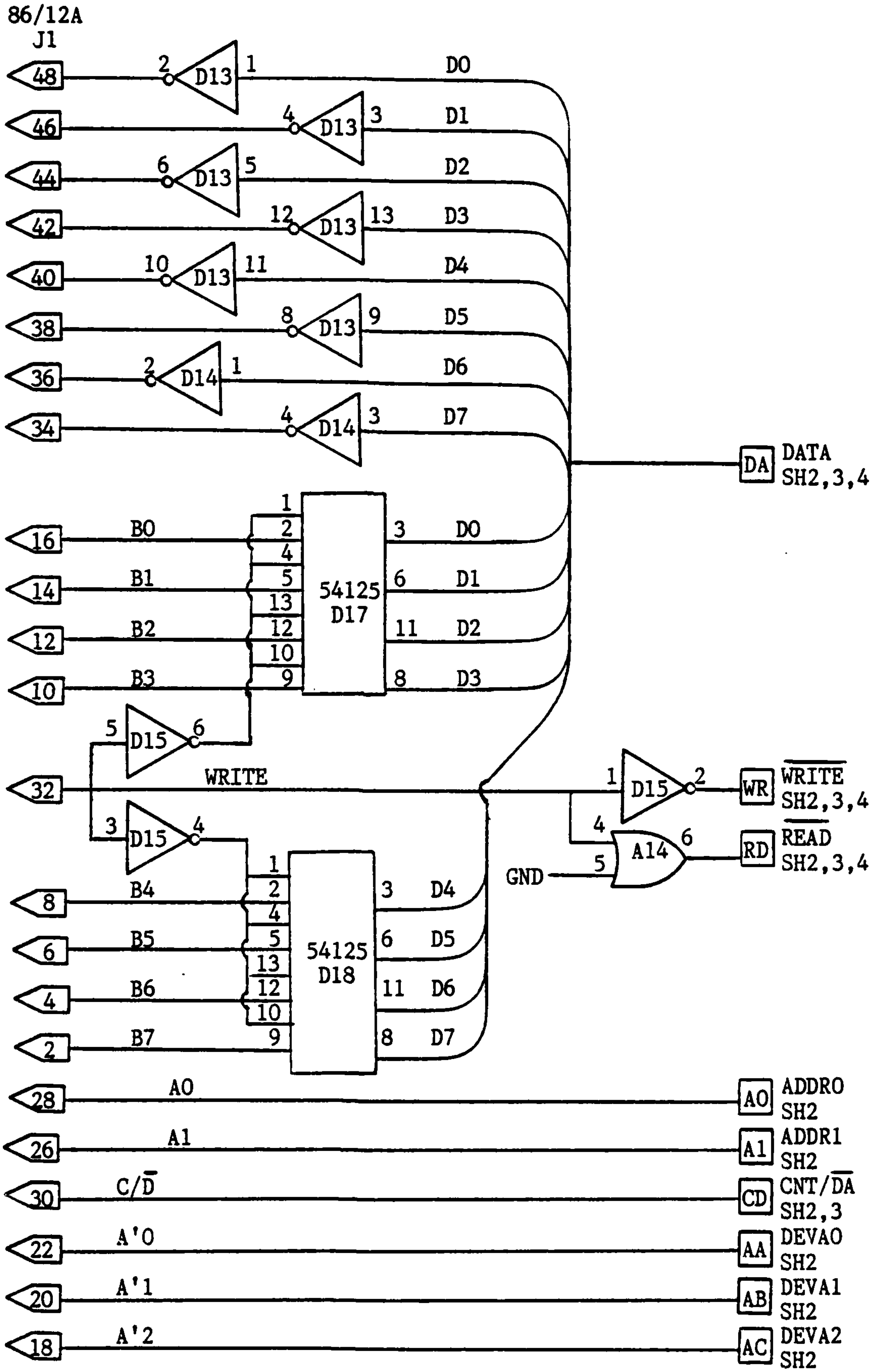
```
/*********************************************************************
*       DATE: 20 OCTOBER 1983                                        *
*       VERSION: 1.2                                                 *
*       NAME: DUMBTERM                                               *
*       MODULE NUMBER: 5                                             *
*       FUNCTION: This function takes the character on the input port *
*                 checks it to see if it is the final one and then   *
*                 puts it on the terminal screen.                    *
*       INPUTS: NONE                                                 *
*       OUTPUTS: NONE                                                *
*       GLOBAL VARIABLES USED: NONE                                  *
*       GLOBAL VARIABLES CHANGED: NONE                               *
*       FILES READ: NONE                                             *
*       FILES WRITTEN: NONE                                          *
*       MODULES CALLED: NONE                                         *
*       CALLING MODULES: MAIN                                        *
*       AUTHOR: William F. Matheson                                  *
*       HISTORY: ORIGINAL                                            *
*********************************************************************/
  dumbterm()

  {

     char  inchar,stat;

     inchar = inportb(PORT);

     outportb(PORT + 3,DTRON); /* let the 86/12A transmit */

     while(inchar != '-') /* check for the prompt character */

     {

        putchar(inchar); /* as long as prompt not received continue
                            to put input character on the screen */

        stat = inportb(PORT + 1);

        while((stat & 02) != 02)

        {
           stat = inportb(PORT + 1);
        }

        inchar = inportb(PORT);
     }

     outportb(PORT + 3, DTROFF); /* turn off DTR when prompt is
                                    received and return to calling
                                    program  */
     putchar(inchar);

     return;
  }
```

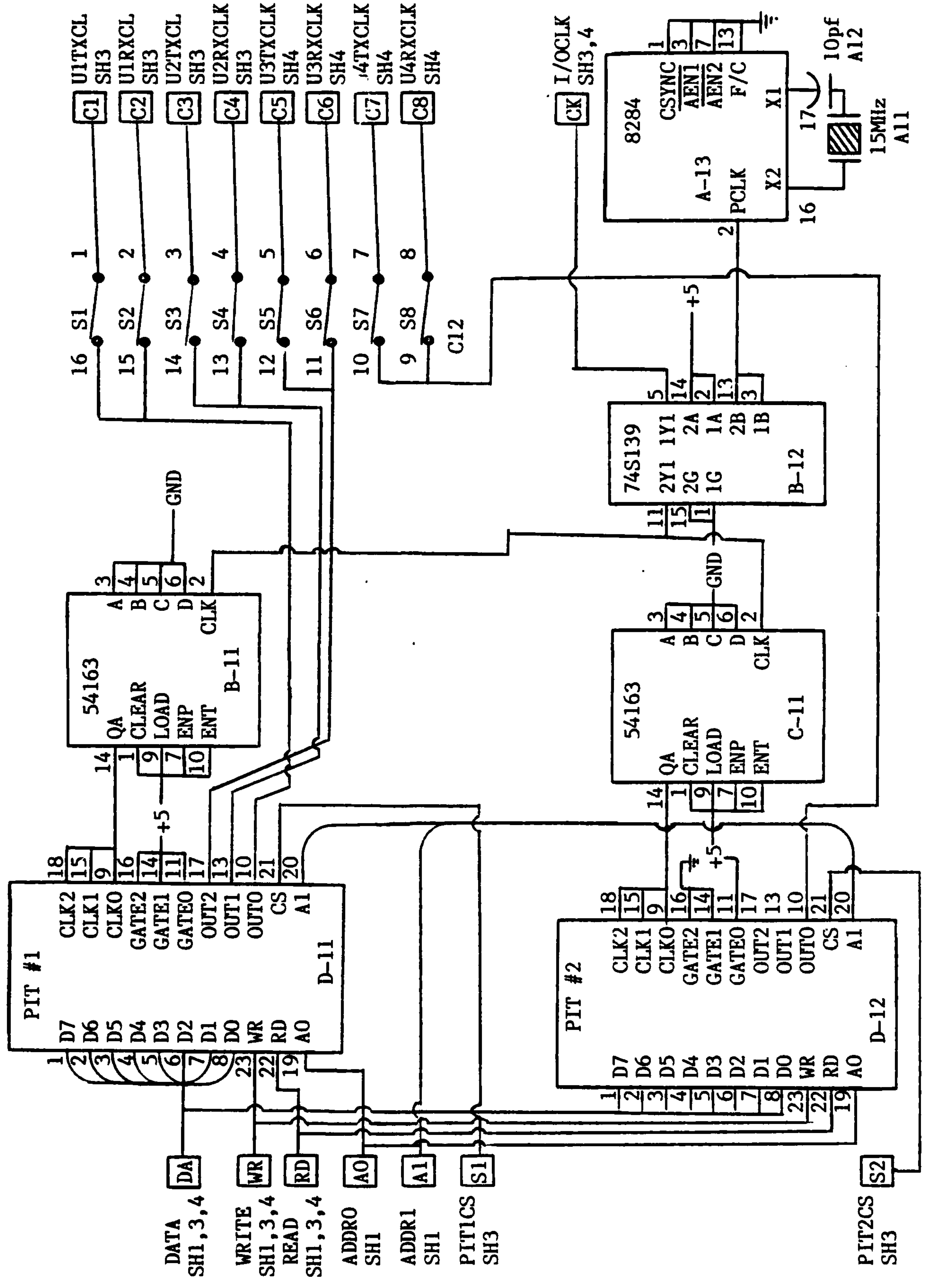
```
/**********************************************************************
*     DATE: 23 OCTOBER 1983                                           *
*     VERSION: 1.0                                                    *
*     NAME: RXCHAR                                                    *
*     MODULE NUMBER: 6                                                *
*     FUNCTION: Gets an input character from the port and checks it   *
*               to see if it is the error code. If it is it puts it   *
*               on the CRT screen and returns FALSE otherwise it      *
*               displays the character on the CRT and returns TRUE.   *
*     INPUTS: none                                                    *
*     OUTPUTS:  TRUE OR FALSE                                         *
*     GLOBAL VARIABLES USED: NONE                                     *
*     GLOBAL VARIABLES CHANGED: NONE                                  *
*     FILES READ: NONE                                                *
*     FILES WRITTEN: NONE                                             *
*     MCDULES CALLED: NONE                                            *
*     CALLING MODULES: MAIN AND LOADER                                *
*     AUTHOR: William F. Matheson                                     *
*     HISTORY: ORIGINAL                                               *
**********************************************************************/

  rxchar()

  {
     char  stat, echo, i;

     outportb(PORT + 3, DTRON);  /*  turn on dtr  */

     for(i = 0; i <= 28; i++); /* SHORT DELAY */
         ;

     outportb(PORT + 3, DTROFF);  /*  turn off the dtr  */

     stat = inportb(PORT + 1); /* get the status byte */

     while((stat & 02) != 02)

        stat = inportb(PORT + 1); /* loop until char in rec buffer */

     echo = inportb(PORT); /* get the received character */

     if(echo == '#') /* is it an error code?  */
     {
        putchar(echo);
        return(FALSE); /* if error return false */
     }
     else
     {
        putchar(echo);
        return(TRUE);  /* otherwise return true  */
     }

  }
```

Appendix D

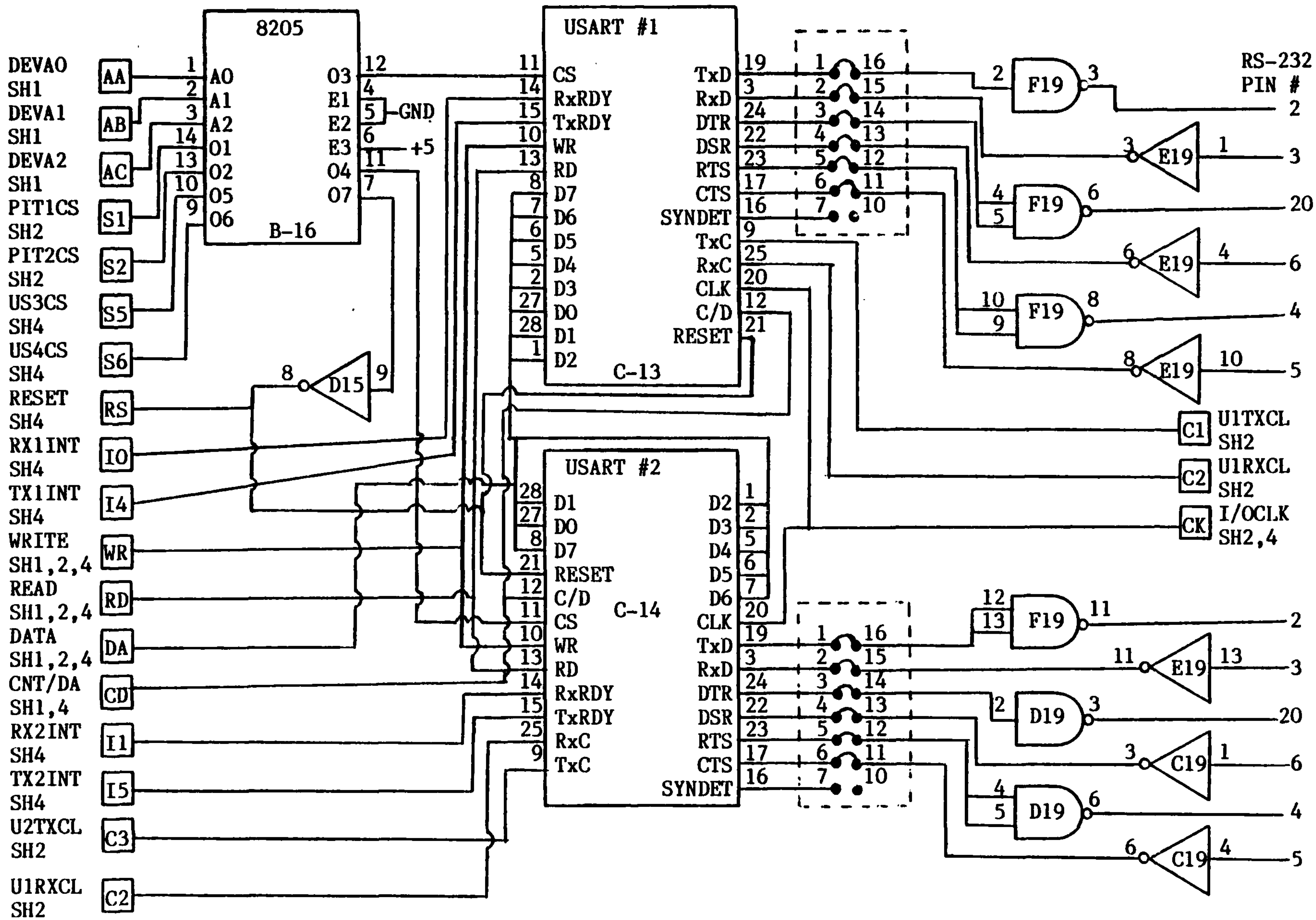
Schematic Diagrams for UNID II

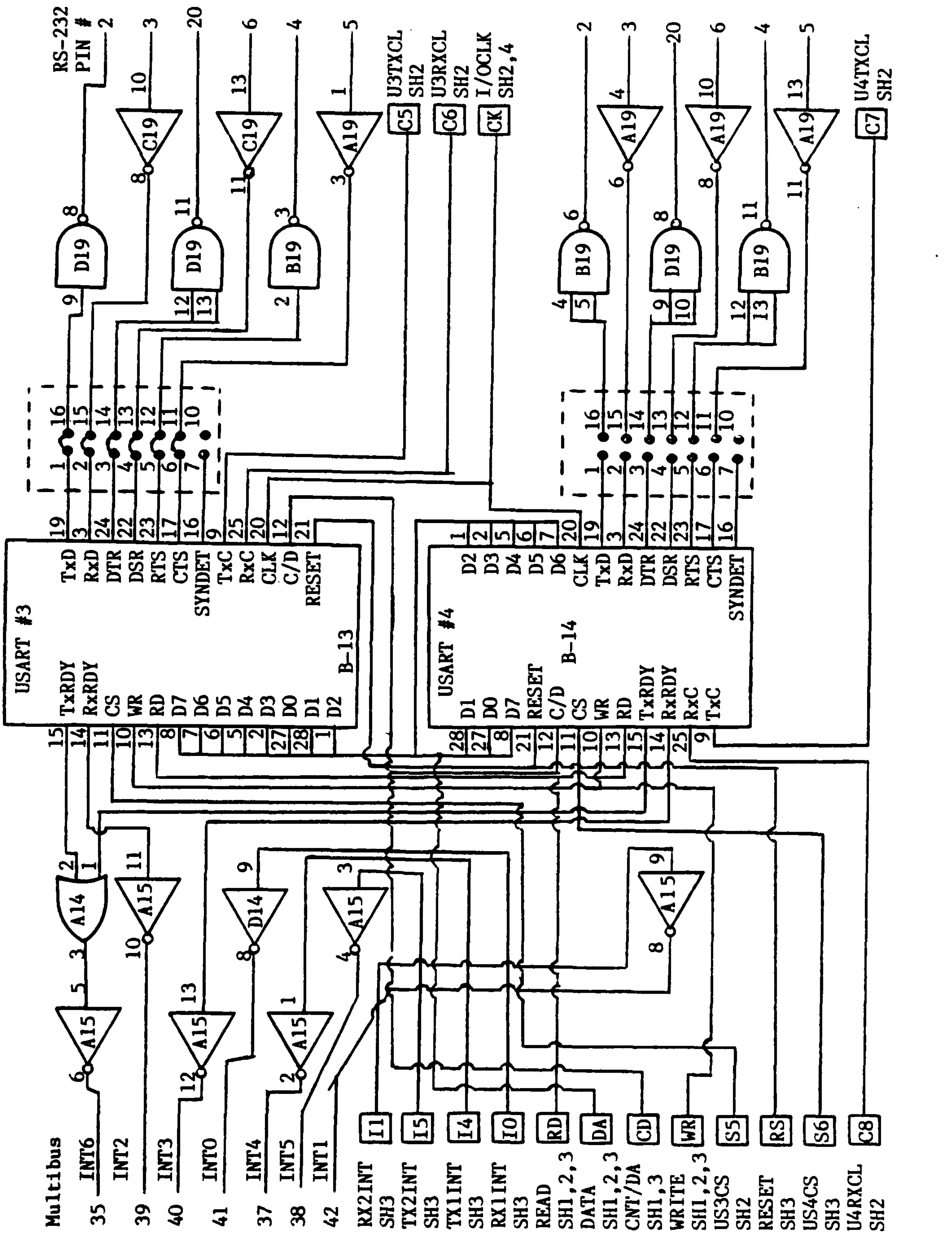
This appendix contains the schematic diagrams for both the local I/O card and the network card. These schematic are an as built version for the local card and the necessary information to construct the network card. The local card schematic is divided into four sections and is shown as Figures D1, D2, D3, and D4. The network card has more components and is divided into five sections shown as Figures D5, D6, D7, D8, and D9. A 0.068 microfarad bypass capacitor should be connected between each chip's VCC and ground. These capacitors are not shown on the schematics.

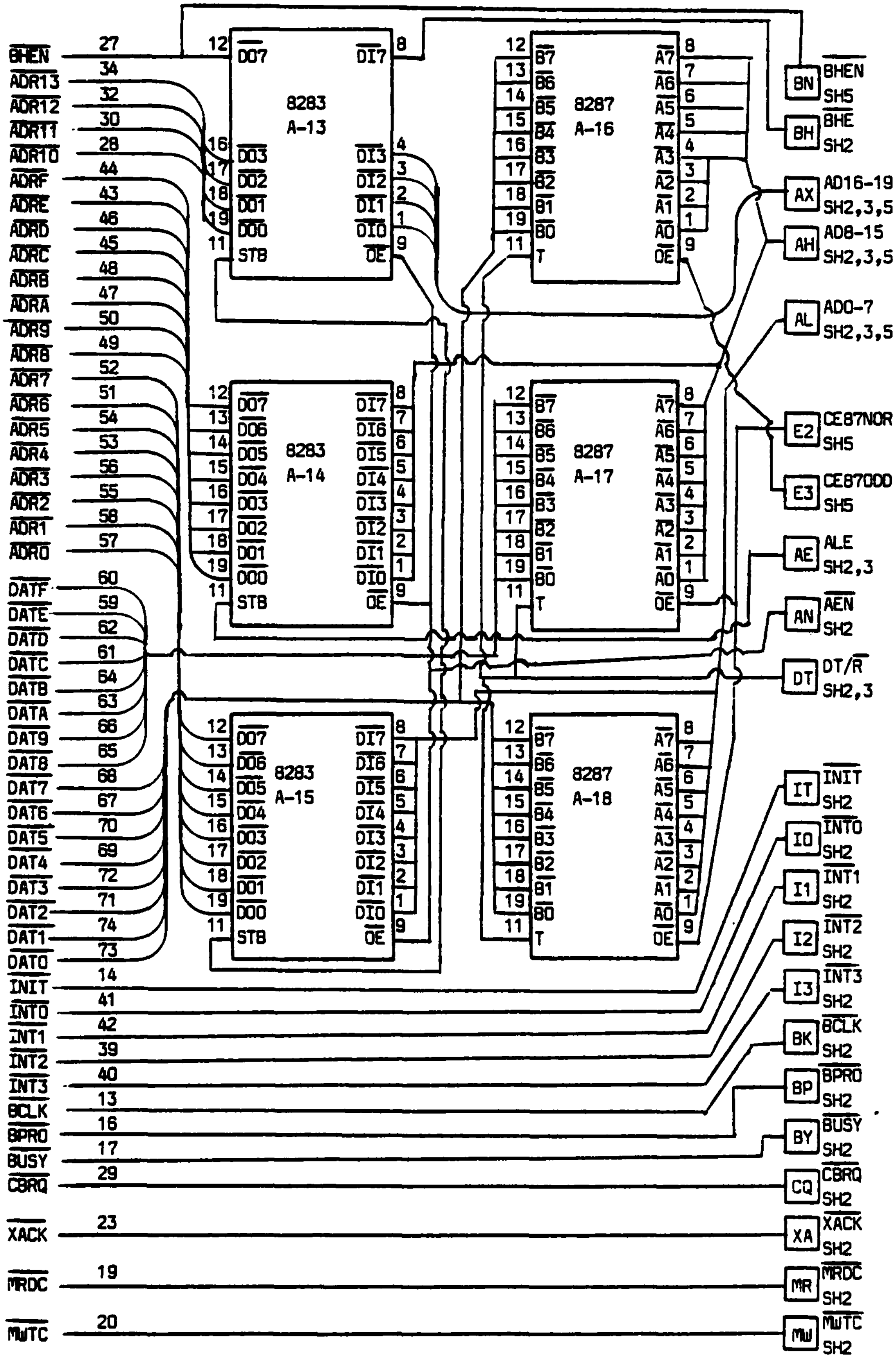

86/12A
J1
48
46
44
42
40
38
36
34
D13
D14
D0
D1
D2
D3
D4
D5
D6
D7
DA
DATA
SH2,3,4
16
14
12
10
B0
B1
B2
B3
54125
D17
32
WRITE
D15
WR
$\overline{\text{WRITE}}$
SH2,3,4
A14
GND
RD
$\overline{\text{READ}}$
SH2,3,4
8
6
4
2
B4
B5
B6
B7
54125
D18
28
A0
ADDR0
SH2
26
A1
ADDR1
SH2
30
C/$\overline{\text{D}}$
CD
CNT/$\overline{\text{DA}}$
SH2,3
22
A'0
AA
DEVA0
SH2
20
A'1
AB
DEVA1
SH2
18
A'2
AC
DEVA2
SH2

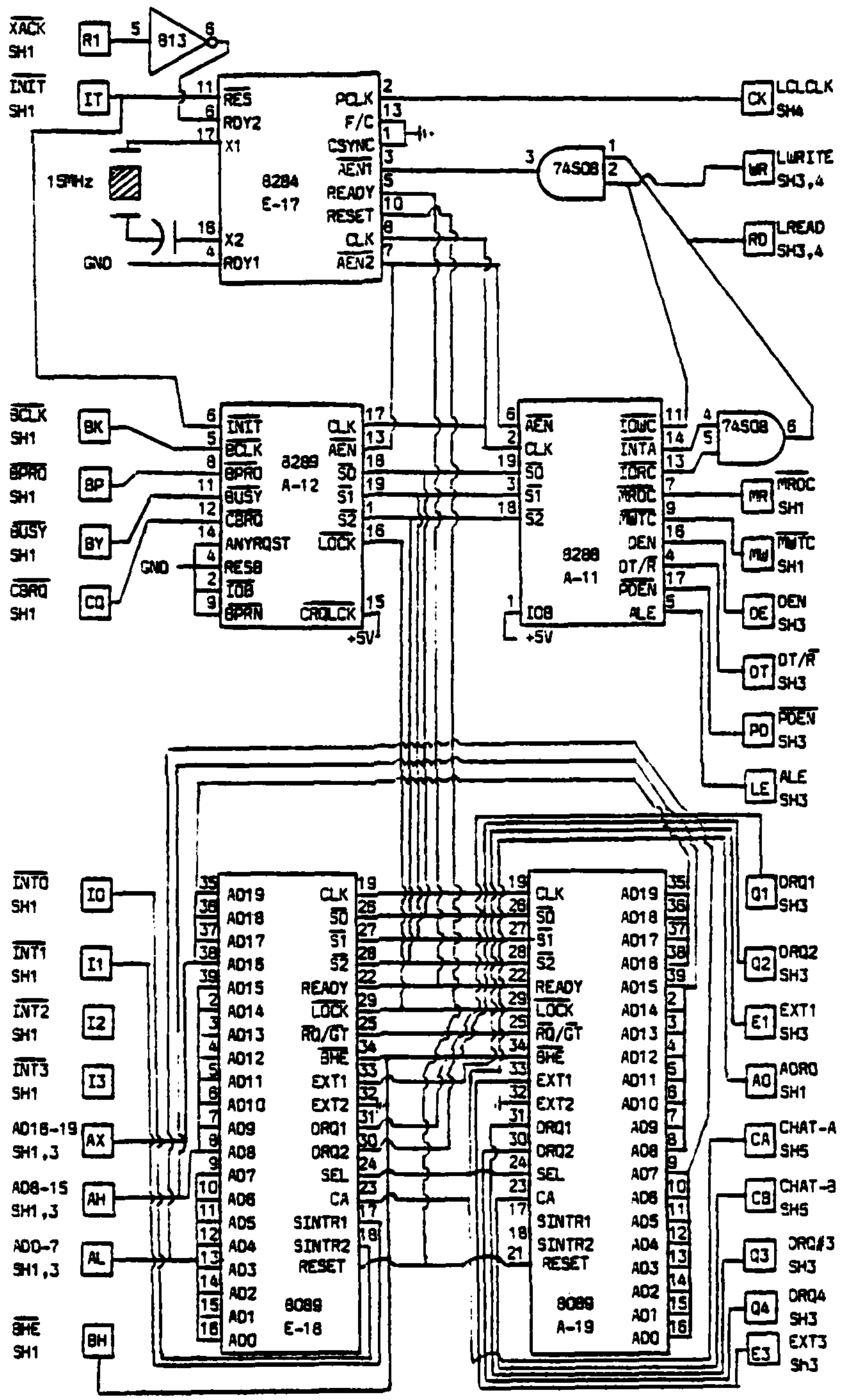


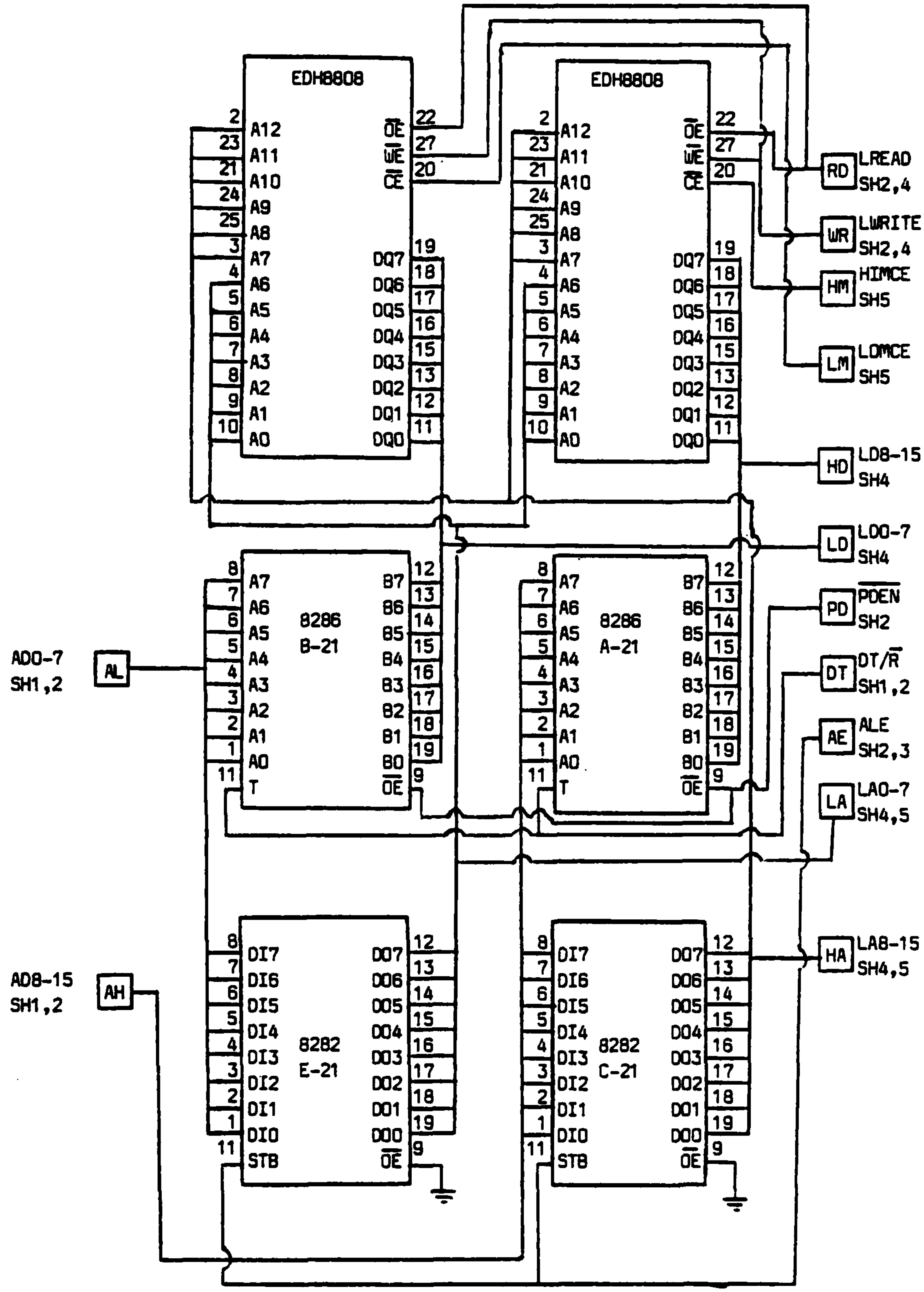
U1TXCL SH3
U1RXCL SH3
U2TXCL SH3
U2RXCLK SH3
U3TXCLK SH4
U3RXCLK SH4
U4TXCLK SH4
U4RXCLK SH4
C1
C2
C3
C4
C5
C6
C7
C8
S1
S2
S3
S4
S5
S6
S7
S8
C12
I/OCLK SH3,4
CK
8284
A-13
CSYNC
AEN1
AEN2
F/C
X1
X2
PCLK
10pf A12
15MHz A11
74S139
B-12
2Y1 1Y1
2G 2A
1G 1A
2B
1B
+5
GND
54163
B-11
QA
CLEAR
LOAD
ENP
ENT
A
B
C
D
CLK
54163
C-11
PIT #1
D-11
CLK2
CLK1
CLK0
GATE2
GATE1
GATE0
OUT2
OUT1
OUT0
CS
A1
D7
D6
D5
D4
D3
D2
D1
D0
WR
RD
A0
PIT #2
D-12
DA
WR
RD
A0
A1
S1
S2
DATA SH1,3,4
WRITE SH1,3,4
READ SH1,3,4
ADDR0 SH1
ADDR1 SH1
PIT1CS SH3
PIT2CS SH3

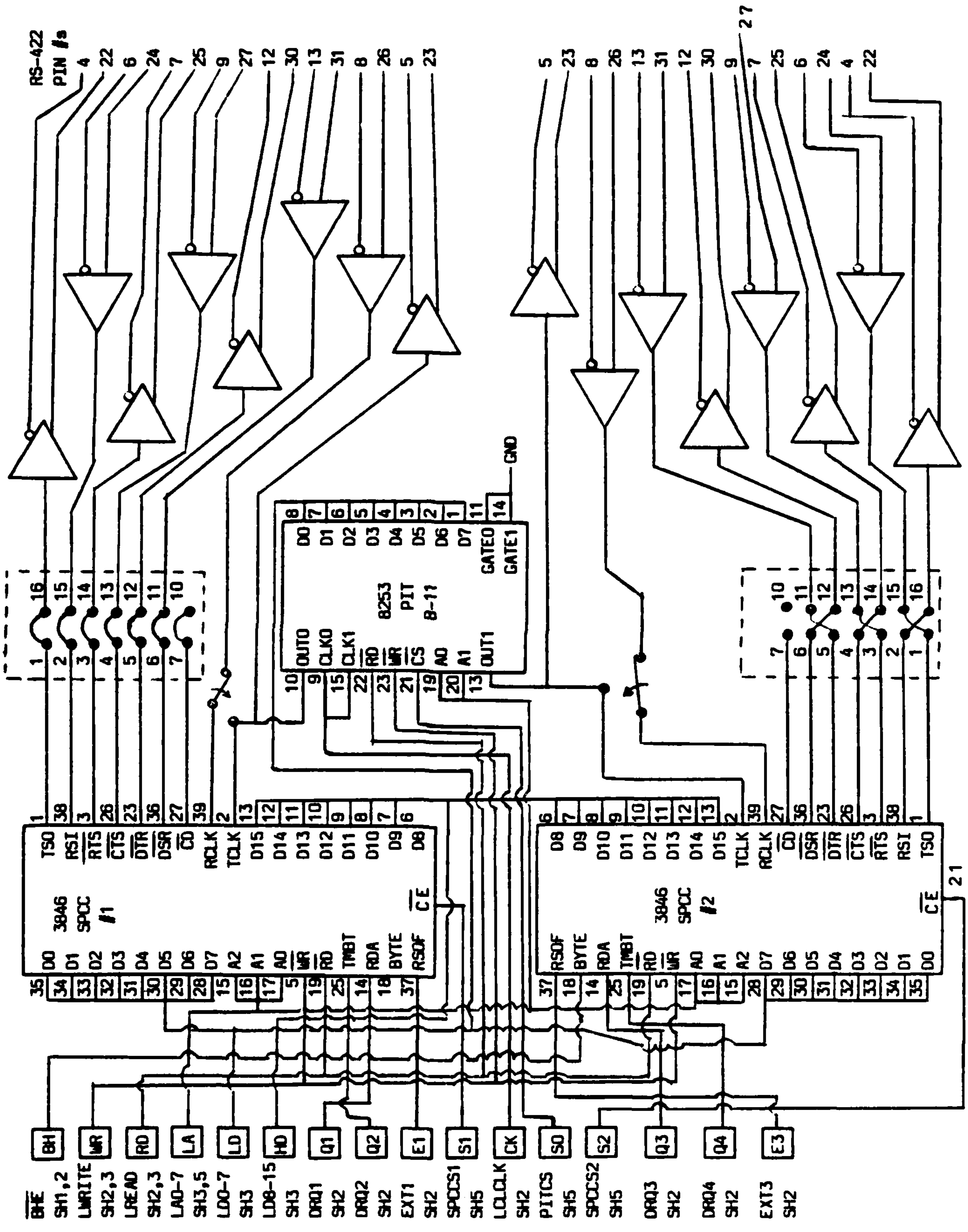

8205
B-16
A0
A1
A2
O1
O2
O5
O6
O3
E1
E2
E3
O4
O7
GND
+5
USART #1
C-13
CS
RxRDY
TxRDY
WR
RD
D7
D6
D5
D4
D3
D0
D1
D2
TxD
RxD
DTR
DSR
RTS
CTS
SYNDET
TxC
RxC
CLK
C/D
RESET
USART #2
C-14
D1
D0
D7
RESET
C/D
CS
WR
RD
RxRDY
TxRDY
RxC
TxC
D2
D3
D4
D5
D6
CLK
TxD
RxD
DTR
DSR
RTS
CTS
SYNDET
D15
F19
E19
D19
C19
RS-232
PIN #
DEVA0
SH1
DEVA1
SH1
DEVA2
SH1
PIT1CS
SH2
PIT2CS
SH2
US3CS
SH4
US4CS
SH4
RESET
SH4
RX1INT
SH4
TX1INT
SH4
WRITE
SH1,2,4
READ
SH1,2,4
DATA
SH1,2,4
CNT/DA
SH1,4
RX2INT
SH4
TX2INT
SH4
U2TXCL
SH2
U1RXCL
SH2
AA
AB
AC
S1
S2
S5
S6
RS
I0
I4
WR
RD
DA
CD
I1
I5
C3
C2
C1
CK
U1TXCL
SH2
U1RXCL
SH2
I/OCLK
SH2,4



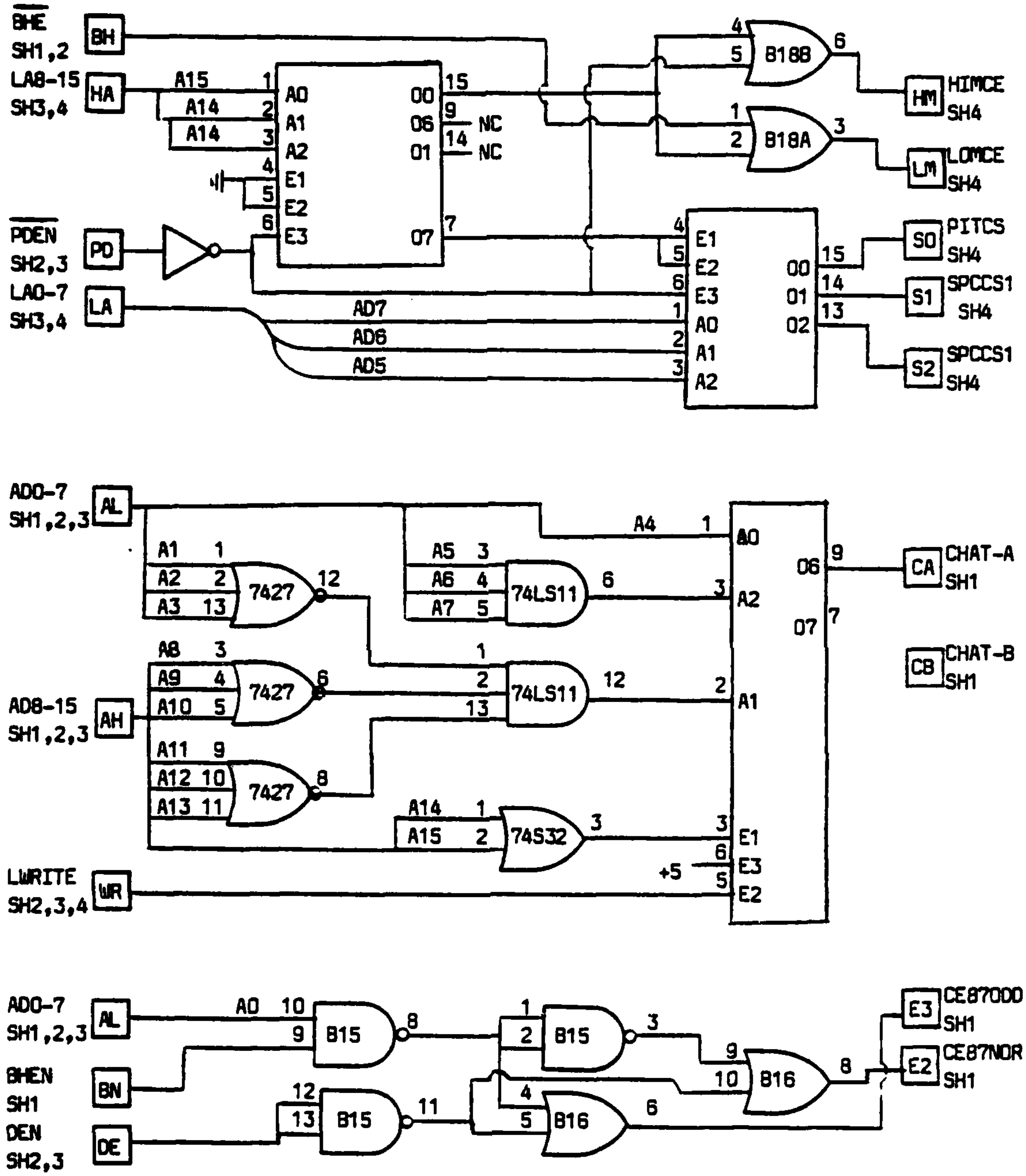

8283 A-13
8287 A-16
8283 A-14
8287 A-17
8283 A-15
8287 A-18
BHEN
ADR13
ADR12
ADR11
ADR10
ADRF
ADRE
ADRD
ADRC
ADRB
ADRA
ADR9
ADR8
ADR7
ADR6
ADR5
ADR4
ADR3
ADR2
ADR1
ADR0
DATF
DATE
DATD
DATC
DATB
DATA
DAT9
DAT8
DAT7
DAT6
DAT5
DAT4
DAT3
DAT2
DAT1
DAT0
INIT
INT0
INT1
INT2
INT3
BCLK
BPRO
BUSY
CBRQ
XACK
MRDC
MWTC
BN BHEN SH5
BH BHE SH2
AX AD16-19 SH2,3,5
AH AD8-15 SH2,3,5
AL AD0-7 SH2,3,5
E2 CE87NOR SH5
E3 CE87ODD SH5
AE ALE SH2,3
AN AEN SH2
DT DT/R SH2,3
IT INIT SH2
I0 INT0 SH2
I1 INT1 SH2
I2 INT2 SH2
I3 INT3 SH2
BK BCLK SH2
BP BPRO SH2
BY BUSY SH2
CQ CBRQ SH2
XA XACK SH2
MR MRDC SH2
MW MWTC SH2


XACK SH1
R1
813
INIT SH1
IT
RES
RDY2
X1
15MHz
X2
RDY1
GND
8284 E-17
PCLK
F/C
CSYNC
AEN1
READY
RESET
CLK
AEN2
CK LCLCLK SH4
74S08
WR LWRITE SH3,4
RD LREAD SH3,4
BCLK SH1
BK
BPRO SH1
BP
BUSY SH1
BY
CBRQ SH1
CQ
INIT
BCLK
BPRO
BUSY
CBRQ
ANYRQST
RESB
IOB
BPRN
8289 A-12
CLK
AEN
S0
S1
S2
LOCK
CRQLCK
+5V
AEN
CLK
S0
S1
S2
IOB
+5V
8288 A-11
IOWC
INTA
IORC
MRDC
MWTC
DEN
DT/R
PDEN
ALE
MR MRDC SH1
MW MWTC SH1
DE DEN SH3
DT DT/R SH3
PD PDEN SH3
LE ALE SH3
INT0 SH1
I0
INT1 SH1
I1
INT2 SH1
I2
INT3 SH1
I3
AD16-19 SH1,3
AX
AD8-15 SH1,3
AH
AD0-7 SH1,3
AL
BHE SH1
BH
8089 E-18
8089 A-19
CLK
S0
S1
S2
READY
LOCK
RQ/GT
BHE
EXT1
EXT2
DRQ1
DRQ2
SEL
CA
SINTR1
SINTR2
RESET
Q1 DRQ1 SH3
Q2 DRQ2 SH3
E1 EXT1 SH3
A0 ADR0 SH1
CA CHAT-A SH5
CB CHAT-B SH5
Q3 DRQ#3 SH3
Q4 DRQ4 SH3
E3 EXT3 Sh3


EDH8808
A12
A11
A10
A9
A8
A7
A6
A5
A4
A3
A2
A1
A0
OE
WE
CE
DQ7
DQ6
DQ5
DQ4
DQ3
DQ2
DQ1
DQ0
RD
LREAD
SH2,4
WR
LWRITE
SH2,4
HM
HIMCE
SH5
LM
LOMCE
SH5
HD
LD8-15
SH4
LD
LD0-7
SH4
8286
B-21
8286
A-21
B7
B6
B5
B4
B3
B2
B1
B0
T
PD
PDEN
SH2
DT
DT/R
SH1,2
AE
ALE
SH2,3
LA
LA0-7
SH4,5
AD0-7
SH1,2
AL
8282
E-21
8282
C-21
DI7
DI6
DI5
DI4
DI3
DI2
DI1
DI0
STB
DO7
DO6
DO5
DO4
DO3
DO2
DO1
DO0
HA
LA8-15
SH4,5
AD8-15
SH1,2
AH



RS-422 PIN #s
8253
PIT
8-11
GND
3846
SPCC
#1
3846
SPCC
#2
BHE
SH1,2
LWRITE
SH2,3
LREAD
SH2,3
LA0-7
SH3,5
LD0-7
SH3
LD8-15
SH3
DRQ1
SH2
DRQ2
SH2
EXT1
SH2
SPCCS1
SH5
LCLCLK
SH2
PITCS
SH5
SPCCS2
SH5
DRQ3
SH2
DRQ4
SH2
EXT3
SH2


BHE
SH1,2
BH
LA8-15
SH3,4
HA
A15
A14
A14
A0
A1
A2
E1
E2
E3
00
06
01
07
NC
NC
B18B
B18A
HM
HIMCE
SH4
LM
LOMCE
SH4
PDEN
SH2,3
PD
LA0-7
SH3,4
LA
AD7
AD6
AD5
S0
PITCS
SH4
S1
SPCCS1
SH4
S2
SPCCS1
SH4
AD0-7
SH1,2,3
AL
A1
A2
A3
7427
A5
A6
A7
74LS11
A4
CA
CHAT-A
SH1
CB
CHAT-B
SH1
AD8-15
SH1,2,3
AH
A8
A9
A10
A11
A12
A13
A14
A15
74S32
LWRITE
SH2,3,4
WR
+5
AD0-7
SH1,2,3
AL
BHEN
SH1
BN
DEN
SH2,3
DE
B15
B16
E3
CE87ODD
SH1
E2
CE87NOR
SH1

Appendix E

Local Input/Output Card

The local I/O card has some special ways of accessing the chips. This appendix has a summary of the control pins assigned to each function and how they work. The pin numbers given are for the control port (po~t C) of the PPI on the iSBC 86/12A card. All control signals are placed on port C of the PPI while port A is used for receive data and port B for the transmit data.

Control Port Pin Assignments

iSBC 86/12A PPI Port C

Data Line	Name	Function
0	Not used	None
1	A'0	This is one of three lines used to select the address of the chips on the I/O card.
2	A'1	Second of the address lines.
3	A'2	Last of the address lines.
4	A1	This is one of two address signals that is routed directly to the 8253 PITs to select the correct register for command instructions.
5	A2	The other address signal for the PITs.
6	C/D	A signal for all four USARTs. This is used to indicate if the information on the bus is to be data or control information.
7	WRITE	Is used to indicate if a write operation is to be done. This signal goes to all chips requiring a write signal. The complement is sent to the read pins. A high indicates a write and a low is for a read.

The specific combination of A0 and A1 used to program the two PITs is shown below.

A0	A1	Function
0	0	Load/Read counter number 0 (for USART 1 if PIT 1 is selected and USART 4 if PIT 2 is selected).
1	0	Load/Read counter number 1 (for USART 2).
0	1	Load/Read counter number 2 (for USART 4).
1	1	Write mode word to PIT when write is high, no action is taken if write is low.

The three A prime signals are decoded by the 8205 to produce chip select signals for the card. A reset to the four USARTs is a special case and care needs to be taken to insure this is not used unless it is needed. The address asignments are shown below.

A'0	A'1	A'2	Chip selected
0	0	0	Not assigned - no chips selected.
1	0	0	PIT number 1.
0	1	0	PIT number 2.
1	1	0	USART number 1.
0	0	1	USART number 2.
1	0	1	USART number 3.
0	1	1	USART number 4.
1	1	1	Reset all four USARTs. This combination of the A prime lines will generate a reset signal to all four USARTs regardless of the other signals. To insure that the USARTs are reset correctly the signal must be maintained for at least 2500 nanoseconds. This is equivalent to 13 clock cycles on the 8086 microprocessor operating at a clock frequency of 5 MHz.

The clock generator (8284) on this card is not used to automatically put the USARTs in the "idle state" on power up so the programmed reset must be used when power is applied. Also, the 8284 on this card can not be synchronized with the one on the iSBC 86/12A and the 8284 on this card can not be reset, and it is not reset when power is applied to the system.

Because of the particular nature of the I/O card there

is a sequence of instructions that must be followed for all read and/or write operations. The two sequences are shown below.

Input (from the I/O card to the iSBC 86/12A)

1. Place the read instruction on port C of the PPI.

2. Use input instruction to accept the data from port A of the PPI.

3. Output an end of read signal to port C (10000000b).

4. If this was a control read rather than a data read then a end of control read signal (11000000b) is send rather than the end of read indicated in step 4>

Output (to the I/O card from the iSBC 86/12A)

1. Use output instruction to place the data byte on port B of the PPI.

2. Use output instruction to place the control byte on port C of the PPI.

3. Write an end of write data (00000000b) to port C.

4. If this was a control write rather than a data write then use the end of write control (01000000b) instead of step 3.

Appendix F

Local Network Software Listing

The program listing that follows was converted from PL/Z to PLM-86 for testing of the local module. The PL/Z program was developed for and tested on the Z-80 based UNID. The conversion was made to insure the two types of UNIDs would operate in the network and to avoid the additional development time if started from scratch.

```
ISIS-II PL/M-86 V2.1 COMPILATION OF MODULE MAIN
OBJECT MODULE PLACED IN :F1:LOCAL.OBJ
COMPILER INVOKED BY:  PLM86 :F1:LOCAL.SRC SMALL


                  $TITLE('LOCAL NETWORK TEST PROGRAM')
                  $XREF OPTIMIZE(2)
                  /*************************************************************************/
                  /**************************************************************************
                   PROLOGUE - MODULE L.MAIN_U1                DATE TRANSLATED: 1 SEP 83
                                                           DATE LAST MODIFIED: 6 NOV 83

                            THE PURPOSE OF THIS MODULE IS TO PROVIDE THE UNID LOCAL
                     OPERATING SYSTEM (L.OS) WITH THE MAIN LINE OF PROCESSING. THE L.OS
                     IS REQUIRED TO INPUT AND OUTPUT DATA FROM EITHER THE FOUR LOCAL
                     CHANNELS OR THE TWO NETWORK CHANNELS.
                            THIS MODULE CONSISTS OF THE MAIN LINE PROCEDURE L.MAIN_U1,
                     AND SUBORDINATE PROCEDURES BUILD_I_PACKET, DET_DEST, LD_TAB_HSKP,
                     SRVC_TAB_HSKP, TRNMIT_PKT, ROUTE_IN, AND ROUTE_OUT.
                            ALL OF THESE ROUTINES ARE CONVERTED FROM THE PL/Z PROGRAMS
                     ORIGINALLY WRITTEN BY CAPT PAUL PHISTER FOR THE Z-80 UNIDS.
                  *************************************************************************/

                  /*************************************************************************/
  1               MAIN:DO;

  2   1             DECLARE   CONCTC        LITERALLY '0D6H', /* MONITOR CTC CMD ADDRESS */
                              CTCR2         LITERALLY '0D4H', /* MONITOR CTC REG2 ADDR   */
                              CONCMD        LITERALLY '0DAH', /* MONITOR USART CMD PORT */
                              CONDAT        LITERALLY '0D8H', /* MONITOR USART DATA PORT */
                              L_RI_DEST_ERR LITERALLY '000H',  /* LOCAL ROUTE_IN ERROR */
                              L_RO_DEST_ERR LITERALLY '01H',   /* LOCAL ROUTE_OUT ERROR */
                              U01DAT        LITERALLY '0C6H',   /* CH-1 USART DATA PORT */
                              U02DAT        LITERALLY '0C8H',   /* CH-2 USART DATA PORT */
                              U03DAT        LITERALLY '0CAH',   /* CH-3 USART DATA PORT */
                              U04DAT        LITERALLY '0CCH';   /* CH-4 USART DATA PORT */

                  /***************************************************************************/
                  /*   MEMORY LOCATIONS FOR THE VECTOR INTERRUPTS                            */
                  /***************************************************************************/
  3   1            DECLARE    INTR$TYPE$32   LITERALLY  '080H',
                              INTR$TYPE$33   LITERALLY  '084H',
                              INTR$TYPE$34   LITERALLY  '088H',
                              INTR$TYPE$35   LITERALLY  '08CH',
                              INTR$TYPE$36   LITERALLY  '090H',
                              INTR$TYPE$37   LITERALLY  '094H',
                              INTR$TYPE$38   LITERALLY  '098H';

                  /*************************************************************************/
                  /*              OFFSET FOR VECTOR                                        */
                  /*************************************************************************/
  4   1            DECLARE   INTR$OFFSET     LITERALLY  '019H';

                  /**************************************************************************/
                  /*  THE PLM86 INTERRUPT STRUCTURE CAUSES AN OFFSET IN THE LOCATION OF THE  */
                  /*  VECTORS. FOR THIS REASON THE VECTOR ADDRESS LOCATION MUST BE CORRECTED */
```

```
                /*  BY A NUMBER OF BYTES. THIS DECLARE SECTION PROVIDES THE MEANS TO SUB-    */
                /*  TRACT A CONSTANT FROM THE VECTOR ADDRESS. THE EXACT AMOUNT WILL BE       */
                /*  DETERMINED BY TESTING THE CODE ON THE UNID.                              */
                /*                                                                           */
                /*****************************************************************************/

 5   1            DECLARE    INTR$IP$32     WORD AT (INTR$TYPE$32),
                             INTR$IP$33     WORD AT (INTR$TYPE$33),
                             INTR$IP$34     WORD AT (INTR$TYPE$34),
                             INTR$IP$35     WORD AT (INTR$TYPE$35),
                             INTR$IP$36     WORD AT (INTR$TYPE$36),
                             INTR$IP$37     WORD AT (INTR$TYPE$37),
                             INTR$IP$38     WORD AT (INTR$TYPE$38);

                                   /* THE FOLLOWING ARE UNID DEFINED VARIABLES */
                                   /* NOTE: THESE VARIABLES MAY CHANGE DEPENDING ON THE
                                            SOFTWARE CONFIGURATION USED WITHIN THE DELNET */

 6   1            DECLARE    DATA_SIZE        LITERALLY '128',    /* NUMBER OF BYTES FROM HOST */
                             PACKET_SIZE      LITERALLY '133',    /* DATA PACKET + HEADER */
                             PACKETS_IN_TABLE LITERALLY '10',
                             STAT_NBR         LITERALLY '20',     /* STATUS ENTRIES IN STATTB */
                             DATA_TABLE_SIZE LITERALLY '1280',    /* PACKET * NBR OF PACKETS */
                             PACKET_TABLE_SIZE LITERALLY '1330', /* PACKET * NBR OF PACKETS */

                                  /* FOLLOWING ARE NETWORK DEFINED VARIABLES */
                                  /* NOTES:1. THIS_UNID_NBR MUST REFLECT WHICH UNID THIS IS.
                                           2. THIS_COUNTRY_CODE MUST REFLECT THE AREA TO WHICH
                                              THIS UNID IS LOCATED.
                                           3. MAX_COUNTRY_CODE WILL INDICATE WHICH COUNTRY CODES
                                              ARE CURRENTLY OPERATIONAL. CC=0000 IS RESERVED FOR
                                              THE DELNET MONITOR.
                                           4. MAX_NETWORK_CODE WILL INDICATE HOW MANY UNIDS ARE
                                              CURRENTLY OPERATIONAL WITHIN A PARTICULAR COUNTRY.
                                           5. FOR DETAILED INFORMATION ON THE ABOVE REFER TO
                                              PHISTER'S THESIS, APPENDIX D.  */
                             THIS_UNID_NBR     LITERALLY '03',  /* UNIQUE ADDRESS FOR THIS UNID */
                             THIS_COUNTRY_CODE LITERALLY '01', /* CC WHERE THIS UNID RESIDES */
                             MAX_COUNTRY_CODE LITERALLY '01',  /* INDICATES COUNTRY_CODES IN USE */
                             MAX_NETWORK_CODE LITERALLY '03';  /* INDICATES UNIDS OPERATIONAL IN NET */

                /*****************************************************************************/
                /*  DEFINITIONS FOR THE LOCAL SERIAL INPUT/OUTPUT CARD AND 86/12 PPI         */
                /*****************************************************************************/

 7   1            DECLARE    CNTRL$8255     LITERALLY '0CEH',  /* 8255 CONTROL PORT ON 86/12 */
                             A$IN           LITERALLY '0C8H',  /* I/O INPUT PORT ADDRESS  */
                             B$OUT          LITERALLY '0CAH',  /* I/O OUTPUT PORT ADDRESS */
                             C$CNTRL        LITERALLY '0CCH',  /* I/O CONTROL PORT ADDRESS */
                             END$WRITE$C    LITERALLY '01000000B', /* CLEARS CONTROL PORT  */
                                                               /* AFTER A CONTROL WRITE   */
                             END$WRITE$D    LITERALLY '00000000B', /* CLEARS CONTROL PORT  */
                                                                   /* AFTER A DATA WRITE   */
                             END$READ$C     LITERALLY '01000000B', /* CLEARS CONTROL PORT  */
                                                                   /* AFTER A CONTROL READ */
                             END$READ$D     LITERALLY '10000000B', /* CLEARS CONTROL PORT  */
                                                                   /* AFTER A DATA READ    */
```

```
                    ICW1$OR$OCW2     LITERALLY 'OCOH',    /* PORT ADDRESS FOR PIC      */
                    ICWS             LITERALLY 'OC2H',    /* PORT ADDRESS FOR PIC MODE */
                    RCV$STATE        LITERALLY '10010110B', /* MODE INSTRUCTION FOR    */
                                                          /* 8251 USART RCV INT      */
                    TRANS$STATE      LITERALLY '10110111B', /* MODE INSTRUCTION FOR    */
                                                          /* 8251 RCV & XMIT INT     */
                    ROTATE$PRIORITY$SET  LITERALLY '10100000B'; /* ROTATES PRIORITY */
                                                          /* FOR EQUAL PRIORITY TO   */
                                                          /* ALL I/O PORTS           */

              /*************************************************************************/
              /*                    INTERRUPT  VECTOR TABLE                            */
              /*************************************************************************/

 8   1          DECLARE  INTR$VECTOR$32  POINTER AT(INTR$TYPE$32),
                         INTR$VECTOR$33  POINTER AT(INTR$TYPE$33),
                         INTR$VECTOR$34  POINTER AT(INTR$TYPE$34),
                         INTR$VECTOR$35  POINTER AT(INTR$TYPE$35),
                         INTR$VECTOR$36  POINTER AT(INTR$TYPE$36),
                         INTR$VECTOR$37  POINTER AT(INTR$TYPE$37),
                         INTR$VECTOR$38  POINTER AT(INTR$TYPE$38);

              /*************************************************************************/
              /*   ADDITIONAL GENERAL DECLARES NEEDED FOR THIS PROGRAM                 */
              /*************************************************************************/

 9   1          DECLARE   BAUD$LSB         BYTE,
                          BAUD$MSB         BYTE,
                          NUM$BYTES$SENT INTEGER,
                          TRANS$1$RDY  BYTE,
                          TRANS$2$RDY  BYTE,
                          TRANS$3$RDY  BYTE,
                          TRANS$4$RDY  BYTE;

10   1       DECLARE      BYTES$SENT$1 BYTE,
                          BYTES$SENT$2 BYTE,
                          BYTES$SENT$3 BYTE,
                          BYTES$SENT$4 BYTE,
                          TRANS$ARRAY$1(128) BYTE,
                          TRANS$ARRAY$2(128) BYTE,
                          TRANS$ARRAY$3(128) BYTE,
                          TRANS$ARRAY$4(128) BYTE;

11   1       DECLARE      DATA$THREE    BYTE,
                          DATA$FOUR     BYTE;
12   1       DECLARE      J             BYTE;

13   1          DECLARE  HSKP_ERR   INTEGER;

              /*************************************************************************/
              /*    DATA TABLES USED IN THIS PROGRAM                                  */
              /*************************************************************************/


14   1       DECLARE     LC01TB (DATA_TABLE_SIZE) BYTE,
                         LC01NS INTEGER,
                         LC01NE INTEGER,
```

```
                           LCO1SZ INTEGER,

                           LCO2TB (DATA_TABLE_SIZE) BYTE,
                           LCO2NS INTEGER,
                           LCO2NE INTEGER,
                           LCO2SZ INTEGER,

                           LCO3TB (DATA_TABLE_SIZE) BYTE,
                           LCO3NS INTEGER,
                           LCO3NE INTEGER,
                           LCO3SZ INTEGER,

                           LCO4TB (DATA_TABLE_SIZE) BYTE,
                           LCO4NS INTEGER,
                           LCO4NE INTEGER,
                           LCO4SZ INTEGER,

                           LCLCTB (DATA_TABLE_SIZE) BYTE,
                           LCLCNS INTEGER,
                           LCLCNE INTEGER,
                           LCLCSZ INTEGER,
                           LCNTTB (PACKET_TABLE_SIZE)  BYTE,
                           LCNTNS INTEGER,
                           LCNTNE INTEGER,
                           LCNTSZ INTEGER,

                           NTLCTB (PACKET_TABLE_SIZE)  BYTE,
                           NTLCNS INTEGER,
                           NTLCNE INTEGER,
                           NTLCSZ INTEGER,

                           STATTB (STAT_NBR) BYTE;

 15   1        DECLARE     TDAADD  POINTER;
 16   1        DECLARE     TPRADD  BYTE  PUBLIC;      /* LOC CHANNEL XMIT PORT ADDRESS  */

                             /*  MISCELLANEOUS DECLARATIONS  */
 17   1        DECLARE     FOREVER   BYTE;
 18   1        DECLARE     BUSYSTATUS  LITERALLY    'OFFH',
                           TRUE        LITERALLY    'OFFH',
                           FALSE       LITERALLY    '00H',
                           NMBR$MSK    LITERALLY    '07H',
                           CR          LITERALLY    '0DH',
                           LF          LITERALLY    '0AH',
                           ESC         LITERALLY    '1BH',
                           E           LITERALLY    '45H',
                           EOT         LITERALLY    '04H';

                             /*  INTERNAL VARIABLES USED IN THIS MODULE  */

 19   1        DECLARE     DESTINATION  INTEGER,   /* DESTINATION OF THE PACKET  */
                           STARTUP_HDR(*) BYTE DATA(CR,LF,
               '                          UNID II #3 LOCAL OS',
               CR,LF,
               '                          VERSION 2 SEP 83 ',
               CR,LF,
               '                             EXECUTING ',EOT,EOT);
```

```
                          /* THE FOLLOWING TEST POINTS ARE USED TO FOLLOW THE DATA WITHIN THE UNID */

 20   1      DECLARE      TP_1(*) BYTE DATA(CR,LF,
              '                   ENTERING INIT_L_TAB PROCEDURE',EOT,EOT);
 21   1      DECLARE      TP_2(*) BYTE DATA(CR,LF,
              '                   ENTERING INIT_U_SHTAB PROCEDURE',EOT,EOT);
 22   1      DECLARE      TP_3(*) BYTE DATA(CR,LF,
              '                   ENTERING INVINT PROCEDURE',EOT,EOT);
 23   1      DECLARE      TP_4(*) BYTE DATA(CR,LF,
              '                   STARTING ROUTE_IN-ROUTE_OUT LOOP',EOT,EOT);
 24   1      DECLARE      TP_5(*) BYTE DATA(CR,LF,
              '                   ENTERING ROUTE_IN PROCEDURE',EOT,EOT);
 25   1      DECLARE      TP_6(*) BYTE DATA(CR,LF,
              '                   DATA LOCATED IN LOCAL CHANNEL-1',EOT,EOT);
 26   1      DECLARE      TP_7(*) BYTE DATA(CR,LF,
              '                   DATA IS LC01TB TO LCLCTB TRANSFER',EOT,EOT);
 27   1      DECLARE      TP_8(*) BYTE DATA(CR,LF,
              '                   DATA IS LC01TB TO LCNTTB TRANSFER',EOT,EOT);
 28   1      DECLARE      TP_9(*) BYTE DATA(CR,LF,
              '                   ERROR OCCURED IN LOCAL CHANNEL-1 IN-PROCESING',EOT,EOT);
 29   1      DECLARE      TP_10(*) BYTE DATA(CR,LF,
              '                   DATA LOCATED IN LOCAL CHANNEL-2',EOT,EOT);
 30   1      DECLARE      TP_11(*) BYTE DATA(CR,LF,
              '                   DATA IS LC02TB TO LCLCTB TRANSFER',EOT,EOT);
 31   1      DECLARE      TP_12(*) BYTE DATA(CR,LF,
              '                   DATA IS LC02TB TO LCNTTB TRANSFER',EOT,EOT);
 32   1      DECLARE      TP_13(*) BYTE DATA(CR,LF,
              '                   ERROR OCCURED IN LOCAL CHANNEL-2 IN PROCESSING',EOT,EOT);
 33   1      DECLARE      TP_14(*) BYTE DATA(CR,LF,
              '                   DATA LOCATED IN LOCAL CHANNEL-3',EOT,EOT);
 34   1      DECLARE      TP_15(*) BYTE DATA(CR,LF,
              '                   DATA IS LC03TB TO LCLCTB TRANSFER',EOT,EOT);
 35   1      DECLARE      TP_16(*) BYTE DATA(CR,LF,
              '                   DATA IS LC03TB TO LCNTTB TRANSFER',EOT,EOT);
 36   1      DECLARE      TP_17(*) BYTE DATA(CR,LF,
              '                   ERROR OCCURED IN LOCAL CHANNEL-3 IN PROCESSING',EOT,EOT);
 37   1      DECLARE      TP_18(*) BYTE DATA(CR,LF,
              '                   DATA LOCATED IN LOCAL CHANNEL-4',EOT,EOT);
 38   1      DECLARE      TP_19(*) BYTE DATA(CR,LF,
              '                   DATA IS LC04TB TO LCLCTB TRANSFER',EOT,EOT);
 39   1      DECLARE      TP_20(*) BYTE DATA(CR,LF,
              '                   DATA IS LC04TB TO LCNTTB TRANSFER',EOT,EOT);
 40   1      DECLARE      TP_21(*) BYTE DATA(CR,LF,
              '                   ERROR OCCURED IN LOCAL CHANNEL-4 IN PROCESSING',EOT,EOT);
 41   1      DECLARE      TP_22(*) BYTE DATA(CR,LF,
              '                   ENTERING ROUTE_OUT PROCEDURE',EOT,EOT);
 42   1      DECLARE      TP_23(*) BYTE DATA(CR,LF,
              '                   OUTGOING DATA IS IS LCLCTB',EOT,EOT);
 43   1      DECLARE      TP_24(*) BYTE DATA(CR,LF,
              '                   DATA IN LCLCTB DESTINED FOR LOCAL CHANNEL-1',EOT,EOT);
 44   1      DECLARE      TP_25(*) BYTE DATA(CR,LF,
              '                   DATA IN LCLCTB DESTINED FOR LOCAL CHANNEL-2',EOT,EOT);
 45   1      DECLARE      TP_26(*) BYTE DATA(CR,LF,
              '                   DATA IN LCLCTB DESTINED FOR LOCAL CHANNEL-3',EOT,EOT);
 46   1      DECLARE      TP_27(*) BYTE DATA(CR,LF,
              '                   DATA LOCATED IN LCLCTB DESTINED FOR LOCAL CHANNEL-4',EOT,EOT);
```

```
47  1       DECLARE      TP_28(*) BYTE DATA(CR,LF,
             '                    ERROR OCCURED IN LCLCTB OUT_PROCESSING',EOT,EOT);
48  1       DECLARE      TP_29(*) BYTE DATA(CR,LF,
             '                    OUTGOING DATA IS IN NTLCTB',EOT,EOT);
49  1       DECLARE      TP_30(*) BYTE DATA(CR,LF,
             '                    DATA IN NTLCTB DESTINED FOR LOCAL CHANNEL-1',EOT,EOT);
50  1       DECLARE      TP_31(*) BYTE DATA(CR,LF,
             '                    DATA IN NTLCTB DESTINED FOR LOCAL CHANNEL-2',EOT,EOT);
51  1       DECLARE      TP_32(*) BYTE DATA(CR,LF,
             '                    DATA IN NTLCTB DESTINED FOR LOCAL CHANNEL-3',EOT,EOT);
52  1       DECLARE      TP_33(*) BYTE DATA(CR,LF,
             '                    DATA IN NTLCTB DESTINED FOR LOCAL CHANNEL-4',EOT,EOT);
53  1       DECLARE      TP_34(*) BYTE DATA(CR,LF,
             '                    ERROR OCCURED IN NTLCTB OUT-PROCESSING',EOT,EOT);
54  1       DECLARE      TP_35(*) BYTE DATA(CR,LF,
             '                    END OF ROUTE_IN-ROUTE_OUT LOOP',EOT,EOT);
55  1       DECLARE      TP_36(*) BYTE DATA(CR,LF,
             '                    HAVE EXITED ROUTE_IN-ROUTE_OUT LOOP',EOT,EOT);

            /***************************************************************************/
            /*   PROCEDURE  DELAY     CAUSES A ONE SECOND DELAY                          */
            /*                                                                           */
            /*        THE PURPOSE OF THIS PROCEDURE IS TO ADD DELAY TO PARTS OF THE     */
            /*        INITIALIZATION PROCEDURE THAT IS TIME DEPENDENT.                  */
            /*                                                                           */
            /*        INPUT - NONE                                                       */
            /*        PROCESSING - USES BUILT IN PROCEDURE IN A LOOP                     */
            /*        OUTPUT - NONE                                                      */
            /*        INTERFACE - CALLED BY INVINT                                       */
            /***************************************************************************/
56  1       DELAY:PROCEDURE PUBLIC;
57  2            DECLARE  I  BYTE;
58  2          DO I=1 TO 40;
59  3             CALL TIME(250); /* TIME IS A BUILT IN FUNCTION OF PLM86 WHICH CAUSES */
                                 /* A DELAY BASED ON THE NUMBER IN PARENS             */
60  3          END;
61  2       END DELAY;

            /****************************************************************************/
            /*                                                                          */
            /* PROCEDURE    INVINT    INITIALIZES THE HARDWARE PORTS                    */
            /*                                                                          */
            /*          THE PURPOSE OF THIS PROCEDURE IS TO INITIALIZE THE VECTOR       */
            /*      TABLES AND ALL THE LOCAL NETWORK PORTS. THIS INCLUDES THE TIMERS,   */
            /*      INTERRUPT CONTROLLER, PARALLEL PORT ON THE ISBC 86/12, AND THE      */
            /*      USARTS.                                                             */
            /*                                                                          */
            /*      INPUT - NONE                                                        */
            /*      PROCESSING - SETS THE VECTOR INTERRUPT TABLE VALUES AND WRITES THE  */
            /*              CONTROLL WORDS TO THE PIC, PIT, PPI, AND USARTS.            */
            /*      OUTPUT - NONE                                                       */
            /*      INTERFACE - CALLED BY MAIN PROGRAM, CALLS DELAY PROCEDURE           */
            /*                                                                          */
            /****************************************************************************/
62  1       INVINT:PROCEDURE;

63  2           DISABLE;   /* PREVENTS ACKNOWLEDGEMENT OF ERRONEOUS INTERRUPTS UNTIL  */
```

```
                              /* PROPER INITIALIZATION OF THE 86/12A ISBC AND THE SCIB   */
                              /* HAS BEEN COMPLETED.  */

                           /*  FILL VECTOR INTERRUPT TABLE WITH PROPER POINTING VECTORS  */

  64   2                 INTR$VECTOR$32 = @SERVICE$RCV$1;
  65   2                 INTR$VECTOR$33 = @SERVICE$RCV$2;
  66   2                 INTR$VECTOR$34 = @SERVICE$RCV$3;
  67   2                 INTR$VECTOR$35 = @SERVICE$RCV$4;
  68   2                 INTR$VECTOR$36 = @SERVICE$TRANS$1;
  69   2                 INTR$VECTOR$37 = @SERVICE$TRANS$2;
  70   2                 INTR$VECTOR$38 = @SERVICE$TRANS$3$AND$4;

            /************************************************************************/
            /*      CORRECT FOR THE VECTOR ADDRESS OFFSET MENTIONED EARLIER          */
            /************************************************************************/
  71   2                 INTR$IP$32 = INTR$IP$32 - INTR$OFFSET;
  72   2                 INTR$IP$33 = INTR$IP$33 - INTR$OFFSET;
  73   2                 INTR$IP$34 = INTR$IP$34 - INTR$OFFSET;
  74   2                 INTR$IP$35 = INTR$IP$35 - INTR$OFFSET;
  75   2                 INTR$IP$36 = INTR$IP$36 - INTR$OFFSET;
  76   2                 INTR$IP$37 = INTR$IP$37 - INTR$OFFSET;
  77   2                 INTR$IP$38 = INTR$IP$38 - INTR$OFFSET;

                          /*  INITIALIZE 8255 PROGRAMMABLE PERIPHERAL INTERFACE (PPI)  */

  78   2                  OUTPUT(CNTRL$8255) = 10010000B;
  79   2                  OUTPUT(C$CNTRL) = END$WRITE$D;

                           /* SET BAUD RATES FOR LOCAL CHANNEL USARTS  */

  80   2                SETBAUD: OUTPUT(B$OUT) = 00110110B;
  81   2                         OUTPUT(C$CNTRL) = 10110010B; /* WRITE, DATA, MODE WORD TO PIT#1 */
  82   2                         OUTPUT(C$CNTRL) = END$WRITE$C;
  83   2                         OUTPUT(B$OUT) = BAUD$LSB;
  84   2                         OUTPUT(C$CNTRL) = 10000010B; /* WRITE, CNTR #0 OF PIT #1 */
  85   2                         OUTPUT(C$CNTRL) = END$WRITE$D;
  86   2                         OUTPUT(B$OUT) = BAUD$MSB;
  87   2                         OUTPUT(C$CNTRL) = 10000010B;
  88   2                         OUTPUT(C$CNTRL) = END$WRITE$D;

                          /*  SET PIT #1, REGISTER #1 FOR USART #2 BAUD RATE  */

  89   2                         OUTPUT(B$OUT) = 01110110B; /* PIT#1, COUNTER #1, 2 BYTES, MODE 3 */
  90   2                         OUTPUT(C$CNTRL) = 10110010B; /* WRITE, DATA, MODE WORD TO PIT #1 */
  91   2                         OUTPUT(C$CNTRL) = END$WRITE$C;
  92   2                         OUTPUT(B$OUT) = BAUD$LSB;
  93   2                         OUTPUT(C$CNTRL) = 10100010B;
  94   2                         OUTPUT(C$CNTRL) = END$WRITE$D;
  95   2                         OUTPUT(B$OUT) = BAUD$MSB;
  96   2                         OUTPUT(C$CNTRL) = 10100010B;
  97   2                         OUTPUT(C$CNTRL) = END$WRITE$D;

                          /* SET PIT #1, REGISTER #2 FOR USART #3 BAUD RATE  */

  98   2                         OUTPUT(B$OUT) = 10110110B;
  99   2                         OUTPUT(C$CNTRL) = 10110010B;
```

```
 100   2                 OUTPUT(C$CNTRL) = END$WRITE$C;
 101   2                 OUTPUT(B$OUT) = BAUD$LSB;
 102   2                 OUTPUT(C$CNTRL) = 10010010B;
 103   2                 OUTPUT(C$CNTRL) = END$WRITE$D;
 104   2                 OUTPUT(B$OUT) = BAUD$MSB;
 105   2                 OUTPUT(C$CNTRL) = 10010010B;
 106   2                 OUTPUT(C$CNTRL) = END$WRITE$D;

              /* SET PIT #2, REGISTER #0 FOR USART #4 BAUD RATE  */

 107   2                 OUTPUT(B$OUT) = 00110110B;
 108   2                 OUTPUT(C$CNTRL) = 10110100B;
 109   2                 OUTPUT(C$CNTRL) = END$WRITE$C;
 110   2                 OUTPUT(B$OUT) = BAUD$LSB;
 111   2                 OUTPUT(C$CNTRL) = 10000100B;
 112   2                 OUTPUT(C$CNTRL) = END$WRITE$D;
 113   2                 OUTPUT(B$OUT) = BAUD$MSB;
 114   2                 OUTPUT(C$CNTRL) = 10000100B;
 115   2                 OUTPUT(C$CNTRL) = END$WRITE$D;

           /*  INITIATE RESET ON ALL USARTS  */

 116   2                 OUTPUT(C$CNTRL) = 00001110B;
 117   2           TIMER:;    /*  CAUSES A 40 SEC DELAY  */
 118   2                 DO J=1 TO 40;
 119   3                    CALL DELAY; /* 1 SECOND DELAY ROUTINE  */
 120   3                 END;
 121   2            NEXT: OUTPUT(C$CNTRL) = END$WRITE$C;
 122   2                 CALL DELAY;

              /*  INITIALIZE THE FOUR USARTS TO: 2 STOP BITS, NO PARITY,  */
              /*     8 BIT CHARACTERS, AND 16X BAUD RATE     */

 123   2                 OUTPUT(B$OUT) = 11001110B;
 124   2                 CALL DELAY;
 125   2                 OUTPUT(C$CNTRL) = 11000110B;  /* USART #1  */
 126   2                 OUTPUT(C$CNTRL) = END$WRITE$C;
 127   2                 CALL DELAY;
 128   2                 OUTPUT(C$CNTRL) = 11001000B;   /* USART #2   */
 129   2                 OUTPUT(C$CNTRL) = END$WRITE$C;
 130   2                 CALL DELAY;
 131   2                 OUTPUT(C$CNTRL) = 11001010B;   /* USART #3   */
 132   2                 OUTPUT(C$CNTRL) = END$WRITE$C;
 133   2                 CALL DELAY;
 134   2                 OUTPUT(C$CNTRL) = 11001100B;   /* USART #4   */
 135   2                 OUTPUT(C$CNTRL) = END$WRITE$C;

               /*  INITIALIZE THE  8259 PROGRAMMABLE INTERRUPT CONTROLLER  */

 136   2                 OUTPUT(ICW1$OR$OCW2) = 00011011B;  /* ICW1  */
                              /* EDGE TRIGGERED MODE, SINGLE PIC, ICW4 NEEDED  */
 137   2                 CALL DELAY;
 138   2                 OUTPUT(ICWS) = 00100000B;  /*  ICW2  */
                               /*  FIRST INTERRUPT TYPE IS 32  */
 139   2                 CALL DELAY;
 140   2                 OUTPUT(ICWS) = 00001101B;  /*  ICW4  */
                               /*  BUFFERED MODE/MASTER, NORMAL EOI, 8086/8088 MODE  */
```

```
141   2                    CALL DELAY;
142   2                    OUTPUT(ICW1$OR$OCW2) = ROTATE$PRIORITY$SET;
                               /*  ROTATE PRIORITY ON NON-SPECIFIC EOI  */
143   2       END INVINT;

              /**********************************************************************/
              /*  PROCEDURE   INIT_L_TAB   INITIALIZES THE LOCAL TABLES              */
              /*                                                                    */
              /*         THIS PROCEDURE SETS THE LOCAL TABLES TO THIER INITIAL VALUES */
              /*     PRIOR TO PROCESSING ANY MESSAGES.                              */
              /*                                                                    */
              /*                                                                    */
              /*     INPUTS - NONE                                                  */
              /*     PROCESSING - PUTS INITIAL VALUES IN THE LOCAL TABLES           */
              /*     OUTPUT - NONE                                                  */
              /*     INTERFACE - CALLED BY THE MAIN PROCEDURE                       */
              /*                                                                    */
              /**********************************************************************/
144   1       INIT_L_TAB:PROCEDURE;
145   2           LC01NS = 0;
146   2           LC01NE = 0;
147   2           LC01SZ = DATA_TABLE_SIZE;

148   2           LC02NS = 0;
149   2           LC02NE = 0;
150   2           LC02SZ = DATA_TABLE_SIZE;

151   2           LC03NS = 0;
152   2           LC03NE = 0;
153   2           LC03SZ = DATA_TABLE_SIZE;

154   2           LC04NS = 0;
155   2           LC04NE = 0;
156   2           LC04SZ = DATA_TABLE_SIZE;

157   2       END INIT_L_TAB;

              /**********************************************************************/
              /*  PROCEDURE  INIT_U_SHTAB    INITIALIZE SHARED TABLES               */
              /*                                                                    */
              /*         THIS PROCEDURE INITIALIZES THE TABLES SHARED BY THE LOCAL  */
              /*      AND NETWORK SOFTWARE.                                         */
              /*                                                                    */
              /*      INPUTS - NONE                                                 */
              /*      PROCESSING - SETS THE SHARED TABLE TO THE INITIAL VALUES NEEDED */
              /*              FOR THE PROPER EXECUTION OF THE PROGRAM.              */
              /*      OUTPUT - NONE                                                 */
              /*      INTERFACE - CALLED BY THE MAIN PROGRAM SEQUENCE.              */
              /*                                                                    */
              /**********************************************************************/
158   1       INIT_U_SHTAB:PROCEDURE;
159   2          DECLARE  IX  WORD;

160   2              LCNTNS =0;
161   2              LCNTNE = 0;
162   2              LCNTSZ = PACKET_TABLE_SIZE;
```

```
 163   2              NTLCNS = 0;
 164   2              NTLCNE = 0;
 165   2              NTLCSZ = PACKET_TABLE_SIZE;

 166   2          TABLE: DO WHILE IX < STAT_NBR;
 167   3                     STATTB(IX) = 0;
 168   3                END TABLE;
 169   2       END INIT_U_SHTAB;

           /************************************************************************/
           /*                                                                      */
           /*  PROCEDURE  SERVICE$RCV$1   RECEIVES DATA ONE CHANNEL ONE            */
           /*                                                                      */
           /*       THE POURPOSE OF THIS PROCEDURE IS TO TAKE A CHARACTER AT A TIME */
           /*       FROM THE RECEIVE PORT ONE AND PUT IT IN THE RECEIVE ONE CHANNEL */
           /*       BUFFER. THIS ROUTINE IS TOTALLY INTERRUPT DRIVEN AND OPERATES   */
           /*       ALL THE TIME AFTER INITIALIZATION.                             */
           /*                                                                      */
           /*       INPUT - NONE (INTERRUPT DRIVEN)                                */
           /*       PROCESSING - MOVES A BYTE OF DATA FROM THE RECEIVE USART TO    */
           /*               MEMORY.                                                */
           /*       OUTPUT - NONE.                                                 */
           /*       INTERFACE - INTERRUPTS SET DURING INITIALIZATION.              */
           /*                                                                      */
           /************************************************************************/
 170   1       SERVICE$RCV$1:PROCEDURE INTERRUPT 32 REENTRANT PUBLIC;
                   /*  ACTIVATED BY INTERRUPT LINE 0   */

 171   2              OUTPUT(C$CNTRL) = 00000110B;
 172   2              LC01TB(LC01NE) = INPUT(A$IN);
 173   2              OUTPUT(C$CNTRL) = END$READ$D;
 174   2              LC01NE = LC01NE + 1;
 175   2              IF(LC01NE >= LC01SZ) THEN
 176   2                 LC01NE = 0;
 177   2       END SERVICE$RCV$1;

           /************************************************************************/
           /*                                                                      */
           /*  PROCEDURE  SERVICE$RCV$2   RECEIVES DATA ON  CHANNEL TWO            */
           /*                                                                      */
           /*       THE POURPOSE OF THIS PROCEDURE IS TO TAKE A CHARACTER AT A TIME */
           /*       FROM THE RECEIVE PORT TWO AND PUT IT IN THE RECEIVE TWO CHANNEL */
           /*       BUFFER. THIS ROUTINE IS TOTALLY INTERRUPT DRIVEN AND OPERATES   */
           /*       ALL THE TIME AFTER INITIALIZATION.                             */
           /*                                                                      */
           /*       INPUT - NONE (INTERRUPT DRIVEN)                                */
           /*       PROCESSING - MOVES A BYTE OF DATA FROM THE RECEIVE USART TO    */
           /*               MEMORY.                                                */
           /*       OUTPUT - NONE.                                                 */
           /*       INTERFACE - INTERRUPTS SET DURING INITIALIZATION.              */
           /*                                                                      */
           /************************************************************************/
 178   1       SERVICE$RCV$2:PROCEDURE INTERRUPT 33 REENTRANT PUBLIC;
                   /*  ACTIVATED BY INTERRUPT LINE 0   */

 179   2              OUTPUT(C$CNTRL) = 00001000B;
 180   2              LC02TB(LC02NE) = INPUT(A$IN);
```

```
181   2              OUTPUT(C$CNTRL) = END$READ$D;
182   2              LC02NE = LC02NE + 1;
183   2              IF(LC02NE >= LC02SZ) THEN
184   2                 LC02NE = 0;
185   2         END SERVICE$RCV$2;

            /************************************************************************/
            /*                                                                      */
            /*  PROCEDURE  SERVICE$RCV$3   RECEIVES DATA ON CHANNEL THREE           */
            /*                                                                      */
            /*        THE POURPOSE OF THIS PROCEDURE IS TO TAKE A CHARACTER AT A TIME */
            /*        FROM THE RECEIVE PORT THREE AND PUT IT IN THE RECEIVE CHANNEL  */
            /*        BUFFER. THIS ROUTINE IS TOTALLY INTERRUPT DRIVEN AND OPERATES  */
            /*        ALL THE TIME AFTER INITIALIZATION.                             */
            /*                                                                      */
            /*        INPUT - NONE (INTERRUPT DRIVEN)                                */
            /*        PROCESSING - MOVES A BYTE OF DATA FROM THE RECEIVE USART TO    */
            /*              MEMORY.                                                  */
            /*        OUTPUT - NONE.                                                 */
            /*        INTERFACE - INTERRUPTS SET DURING INITIALIZATION.              */
            /*                                                                      */
            /************************************************************************/
186   1         SERVICE$RCV$3:PROCEDURE INTERRUPT 34 REENTRANT PUBLIC;
                      /*  ACTIVATED BY INTERRUPT LINE 2   */

187   2              OUTPUT(C$CNTRL) = 00000101OB;
188   2              LC03TB(LC03NE) = INPUT(A$IN);
189   2              OUTPUT(C$CNTRL) = END$READ$D;
190   2              LC03NE = LC03NE + 1;
191   2              IF(LC03NE >= LC03SZ) THEN
192   2                 LC03NE = 0;
193   2         END SERVICE$RCV$3;

            /************************************************************************/
            /*                                                                      */
            /*  PROCEDURE  SERVICE$RCV$4   RECEIVES DATA ON  CHANNEL FOUR           */
            /*                                                                      */
            /*        THE POURPOSE OF THIS PROCEDURE IS TO TAKE A CHARACTER AT A TIME */
            /*        FROM THE RECEIVE PORT FOUR AND PUT IT IN THE RECEIVE CHANNEL   */
            /*        BUFFER. THIS ROUTINE IS TOTALLY INTERRUPT DRIVEN AND OPERATES  */
            /*        ALL THE TIME AFTER INITIALIZATION.                             */
            /*                                                                      */
            /*        INPUT - NONE (INTERRUPT DRIVEN)                                */
            /*        PROCESSING - MOVES A BYTE OF DATA FROM THE RECEIVE USART TO    */
            /*              MEMORY.                                                  */
            /*        OUTPUT - NONE.                                                 */
            /*        INTERFACE - INTERRUPTS SET DURING INITIALIZATION.              */
            /*                                                                      */
            /************************************************************************/
194   1         SERVICE$RCV$4:PROCEDURE INTERRUPT 35 REENTRANT PUBLIC;
                      /*  ACTIVATED BY INTERRUPT LINE 3   */

195   2              OUTPUT(C$CNTRL) = 00000110OB;
196   2              LC04TB(LC04NE) = INPUT(A$IN);
197   2              OUTPUT(C$CNTRL) = END$READ$D;
198   2              LC04NE = LC04NE + 1;
199   2              IF(LC04NE >= LC04SZ) THEN
```

```
                    LCO4NE = 0;
        END SERVICE$RCV$4;

        /*************************************************************************/
        /*  PROCEDURE   SERVICE$TRANS$1   SENDS DATA OUT CHANNEL ONE            */
        /*                                                                       */
        /*       THE PURPOSE OF THIS PROCEDURE IS TO SEND A PACKET OF DATA OUT   */
        /*       LOCAL CHANNEL ONE. A SINGLE BYTE IS TRANSMITTED EACH TIME AN    */
        /*       INTERRUPT IS GENERATED BY USART ONE ON THE TRANSMIT SIDE.       */
        /*                                                                       */
        /*       INPUT - NONE (INTERRUPT DRIVEN)                                 */
        /*       PROCESSING - SENDS A BYTE OF DATA FROM THE TRANSMIT ARRAY TO    */
        /*               THE DATA PORT AND THEN WRITES THE CONTROL INFORMATION   */
        /*               TO THE CONTROL PORT. WHEN MESSAGE IS DONE IT RESETS     */
        /*               THE TRANSMIT INTERRUPT AND SETS TRANS$1$RDY TO TRUE.    */
        /*        OUTPUT - NONE.                                                 */
        /*        INTERFACE - CALLED BY MAIN PROCEDURE.                          */
        /*                                                                       */
        /*************************************************************************/
        SERVICE$TRANS$1:PROCEDURE INTERRUPT 36 REENTRANT PUBLIC;
           IF(BYTES$SENT$1  <= DATA_SIZE) THEN
              DO;
                 OUTPUT(B$OUT) = TRANS$ARRAY$1(BYTES$SENT$1);
                 OUTPUT(C$CNTRL) = 1000$0110B;
                 OUTPUT(C$CNTRL) = END$WRITE$D;
                 BYTES$SENT$1 = BYTES$SENT$1 + 1;
               END;

            ELSE    /* RESET TRANSMIT INTERRUPT      */
               DO;
                  OUTPUT(B$OUT) = RCV$STATE;
                  OUTPUT(C$CNTRL) = 1100$0110B;
                  OUTPUT(C$CNTRL) = END$WRITE$D;
                  BYTES$SENT$1 = 0;
                  TRANS$1$RDY = TRUE;
                END;
        END SERVICE$TRANS$1;


        /*************************************************************************/
        /*  PROCEDURE   SERVICE$TRANS$2   SENDS DATA OUT CHANNEL TWO            */
        /*                                                                       */
        /*       THE PURPOSE OF THIS PROCEDURE IS TO SEND A PACKET OF DATA OUT   */
        /*       LOCAL CHANNEL TWO. A SINGLE BYTE IS TRANSMITTED EACH TIME AN    */
        /*       INTERRUPT IS GENERATED BY USART TWO ON THE TRANSMIT SIDE.       */
        /*                                                                       */
        /*       INPUT - NONE (INTERRUPT DRIVEN)                                 */
        /*       PROCESSING - SENDS A BYTE OF DATA FROM THE TRANSMIT ARRAY TO    */
        /*               THE DATA PORT AND THEN WRITES THE CONTROL INFORMATION   */
        /*               TO THE CONTROL PORT. WHEN MESSAGE IS DONE IT RESETS     */
        /*               THE TRANSMIT INTERRUPT AND SETS TRANS$1$RDY TO TRUE.    */
        /*        OUTPUT - NONE.                                                 */
        /*        INTERFACE - CALLED BY MAIN PROCEDURE.                          */
        /*                                                                       */
        /*************************************************************************/
        SERVICE$TRANS$2: PROCEDURE INTERRUPT 37 REENTRANT PUBLIC;
```

```
 219   2          IF(BYTES$SENT$2 <= DATA_SIZE) THEN
 220   2            DO;
 221   3              OUTPUT(B$OUT) = TRANS$ARRAY$1(BYTES$SENT$2);
 222   3              OUTPUT(C$CNTRL) = 1000$1000B;
 223   3              OUTPUT(C$CNTRL) = END$WRITE$D;
 224   3              BYTES$SENT$2 = BYTES$SENT$2 + 1;
 225   3            END;

                 ELSE    /* RESET TRANSMIT INTERRUPT       */
 226   2            DO;
 227   3              OUTPUT(B$OUT) = RCV$STATE;
 228   3              OUTPUT(C$CNTRL) = 1100$1000B;
 229   3              OUTPUT(C$CNTRL) = END$WRITE$D;
 230   3              BYTES$SENT$2 = 0;
 231   3              TRANS$2$RDY = TRUE;
 232   3            END;
 233   2        END SERVICE$TRANS$2;

          /***************************************************************************/
          /*  PROCEDURE   SERVICE$TRANS$3&4    SENDS DATA OUT CHANNEL THREE AND FOUR */
          /*                                                                         */
          /*        THE PURPOSE OF THIS PROCEDURE IS TO SEND A PACKET OF DATA OUT    */
          /*        LOCAL CHANNEL ONE. A SINGLE BYTE IS TRANSMITTED EACH TIME AN     */
          /*        INTERRUPT IS GENERATED BY USART ONE ON THE TRANSMIT SIDE.        */
          /*                                                                         */
          /*        INPUT - NONE (INTERRUPT DRIVEN)                                  */
          /*        PROCESSING - SENDS A BYTE OF DATA FROM THE TRANSMIT ARRAY TO     */
          /*               THE DATA PORT AND THEN WRITES THE CONTROL INFORMATION     */
          /*               TO THE CONTROL PORT. WHEN MESSAGE IS DONE IT RESETS       */
          /*               THE TRANSMIT INTERRUPT AND SETS TRANS$1$RDY TO TRUE.      */
          /*         OUTPUT - NONE.                                                  */
          /*         INTERFACE - CALLED BY MAIN PROCEDURE.                           */
          /*                                                                         */
          /***************************************************************************/
 234   1        SERVICE$TRANS$3$AND$4:PROCEDURE INTERRUPT 38 REENTRANT PUBLIC;
 235   2            DECLARE STATUS BYTE;

 236   2            IF DATA$THREE THEN DO;
 238   3              IF (BYTES$SENT$3 <= DATA_SIZE) THEN DO;
 240   4                OUTPUT(C$CNTRL) = 01001010B;
 241   4                STATUS = INPUT(A$IN);
 242   4                OUTPUT(C$CNTRL) = END$READ$C;
 243   4                IF (STATUS OR 11111011B) THEN DO;
 245   5                  OUTPUT(B$OUT) = TRANS$ARRAY$3(BYTES$SENT$3);
 246   5                  OUTPUT(C$CNTRL) = 10001010B;
 247   5                  OUTPUT(C$CNTRL) = END$WRITE$D;
 248   5                  BYTES$SENT$3 = BYTES$SENT$3 + 1;
 249   5                END;
 250   4              END;
 251   3              ELSE DO;
 252   4                OUTPUT(B$OUT) = RCV$STATE;
 253   4                OUTPUT(C$CNTRL) = 10001010B;
 254   4                OUTPUT(C$CNTRL) = END$WRITE$D;
 255   4                BYTES$SENT$3 = 0;
 256   4                TRANS$3$RDY = TRUE;
 257   4              END;
 258   3            END;
```

```
259   2          IF DATA$FOUR THEN DO;
261   3             IF(BYTES$SENT$4 <= DATA_SIZE) THEN DO;
263   4                OUTPUT(C$CNTRL) = 01001100B;
264   4                STATUS = INPUT(A$IN);
265   4                OUTPUT(C$CNTRL) = END$READ$C;
266   4                IF(STATUS OR 11111011B) THEN DO;
268   5                   OUTPUT(B$OUT) = TRANS$ARRAY$4(BYTES$SENT$4);
269   5                   OUTPUT(C$CNTRL) = 10001100B;
270   5                   OUTPUT(C$CNTRL) = END$WRITE$D;
271   5                   BYTES$SENT$4 = BYTES$SENT$4 + 1;
272   5                 END;
273   4               END;
274   3               ELSE DO;
275   4                 OUTPUT(B$OUT) = RCV$STATE;
276   4                 OUTPUT(C$CNTRL) = 10001100B;
277   4                 OUTPUT(C$CNTRL) = END$WRITE$D;
278   4                 BYTES$SENT$4 = 0;
279   4                 TRANS$4$RDY = TRUE;
280   4               END;
281   3             END;
282   2      END SERVICE$TRANS$3$AND$4;

        /*************************************************************************/
        /*  PROCEDURE   SNDSEQ   SENDS DATA TO LOCAL MONITOR FOR TESTING          */
        /*                                                                       */
        /*              THIS PROCEDURE TAKES A MESSAGE STRING AND OUTPUTS IT TO  */
        /*         THE LOCAL MONITOR ATTACHED TO THE 86/12 CARD.                 */
        /*                                                                       */
        /*         INPUT - A POINTER TO THE MESSAGE LOCATION IN MEMORY AND THE   */
        /*                 NUMBER OF BYTES TO BE SENT ARE INPUT.                 */
        /*         PROCESSING - THIS PROCEDURE CHECKS THE OUTPUT BUFFER STATUS   */
        /*                 IN A LOOP UNTIL THE BUFFER IS EMPTY. IT PLACES ONE    */
        /*                 BYTE AT A TIME IN THE OUTPUT USART UNTIL THE MESSAGE  */
        /*                 IS DONE.                                              */
        /*         OUTPUT - MESSAGE TO THE USART CHANNEL.                        */
        /*         INTERFACE - THIS PROCEDURE IS CALLED BY THE FOLLOWING PROCEDURES: */
        /*                       ROUTE_IN, ROUTE_OUT, AND MAIN.                  */
        /*                                                                       */
        /*************************************************************************/
283   1     SNDSEQ: PROCEDURE(MSG, TOTAL) REENTRANT;
284   2            DECLARE  MSG  POINTER,CHAROUT BASED MSG(1) BYTE;
285   2            DECLARE  TOTAL  BYTE;
286   2            DECLARE  STATUS  BYTE,
                            COUNT   BYTE;
287   2            COUNT = 0;
288   2            OUTPUT(CONCMD) = 10110011B;
289   2        LOOP: DO WHILE COUNT < TOTAL;
290   3             STATUS = INPUT(CONCMD);
291   3      INLOOP:    DO WHILE(STATUS OR 11111110B);
292   4                   STATUS = INPUT(CONCMD);
293   4               END INLOOP;
294   3             OUTPUT(CONDAT) = CHAROUT(COUNT);
295   3             COUNT = COUNT + 1;
296   3           END LOOP;
297   2      END SNDSEQ;
```

```
                /***************************************************************************
                 PROCEDURE DET_DEST_ONE   DETERMINES THE DESTINATION OF DATA FROM LOCAL HOST

                     THE PURPOSE OFTHIS PROCEDURE IS TO DETERMINE THE DESTINATION OF DATA
                      COMING FROM A HOST CONNECTED TO LOCAL CHANNEL-1.

                     INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING THE TABLE
                             LOCATION OF THE PACKET TO BE EVALUATED AND A LOCATION FOR THE
                             DESTINATION TO BE PLACED..

                     PROCESSING - THE PROCEDURE FIRST EXTRACTS THE CONTROL_CODE FROM THE INCOMMING
                             DATA TO DETERMINE WHICH ROUTING SCHEME IS USED. THE COUNTRY_CODE
                             AND NETWORK_CODE ARE THEN EXTRACTED TO DETERMINE THE DATA'S
                             DESTINATION. IF THE DATA IS DESTINED FOR THE NETWORK SIDE OF THE
                             UNID THEN THE HOST_CODE IS ALSO EXTRACTED SO THAT THE
                             DESTINATION_ADDRESS CAN BE DETERMINED.

                     OUTPUT - THIS PROCEDURE PLACES THE DESTINATION ('LN' OR 'LL') IN MEMORY
                             FOR THE PASSING ROUTINE AND IT RETURNS
                             THE DESTINATION_ADDRESS. AN EXAMPLE OF A DESTINATION_ADDRESS
                             IS 21, WHICH INDICATES UNID=2 AND CHANNEL=1.

                     INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE ROUTE_IN.

                     NOTES: 1. BYTE AND OFH WILL MASK OUT THE UPPER 4-BITS.
                            2. BYTE AND OFOH WILL MASK OUT THE LOWER 4-BITS.
                            3. BYTE / 01H WILL RETURN A VALUE 0-15 FROM THE LOWER 4-BITS.
                            4. BYTE / 10H WILL RETURN A VALUE 0-15 FROM THE UPPER 4-BITS.

                ***************************************************************************/

298   1         DET_DEST_ONE: PROCEDURE BYTE REENTRANT;

299   2           DECLARE BITS BYTE,
                          NEWBITS BYTE,
                          LASTBITS BYTE,
                          CONTROL_CODE BYTE,
                          COUNTRY_CODE BYTE,
                          NETWORK_CODE BYTE,
                          HOST_CODE WORD,
                          DESTINATION_ADDRESS BYTE;

300   2           HOST_CODE = 0;
301   2           CONTROL_CODE = LC01TB(LC01NS + 16) AND OFOH;
302   2           IF CONTROL_CODE = 00 THEN DO;
304   3             COUNTRY_CODE = LC01TB(LC01NS + 16) AND OFH;
305   3             IF COUNTRY_CODE <= MAX_COUNTRY_CODE THEN DO;
307   4               NETWORK_CODE = LC01TB(LC01NS + 17) AND OFOH;
308   4               IF (NETWORK_CODE /10H) <= MAX_NETWORK_CODE THEN DO;
310   5                 IF (COUNTRY_CODE <> THIS_COUNTRY_CODE) OR ((NETWORK_CODE /10H)<>
                        THIS_UNID_NBR) THEN DO;
312   6                   DESTINATION = 6;
313   6                   BITS = ROL(LC01TB(LC01NS + 17),4);
314   6                   NEWBITS = BITS AND OFOH;
315   6                   BITS = ROR(LC01TB(LC01NS + 18),4);
316   6                   LASTBITS = BITS AND OFH;
317   6                   HOST_CODE = DOUBLE(BITS OR LASTBITS);
318   6                   IF (HOST_CODE >=0) AND (HOST_CODE <= 63) THEN
```

```
319   6                   DESTINATION_ADDRESS = NETWORK_CODE OR 01H;
320   6               ELSE IF (HOST_CODE >=64) AND (HOST_CODE <= 127) THEN
321   6                   DESTINATION_ADDRESS = NETWORK_CODE OR 02H;
322   6               ELSE IF (HOST_CODE >= 128) AND (HOST_CODE <= 191) THEN
323   6                   DESTINATION_ADDRESS = NETWORK_CODE OR 03H;
324   6               ELSE IF (HOST_CODE >= 192) AND (HOST_CODE <=255) THEN
325   6                   DESTINATION_ADDRESS = NETWORK_CODE OR 04H;
326   6               ELSE DO;
327   7                   DESTINATION = 5;
328   7               END;
329   6             END;
330   5             ELSE DO;
331   6                 DESTINATION = 8;
332   6                 STATTB(04) = STATTB(04) + 1;
333   6                 STATTB(00) = STATTB(00) + 1;
334   6             END;
335   5           END;
336   4           ELSE DO;
337   5               DESTINATION = 8;
338   5               STATTB(03) = STATTB(03) + 1;
339   5               STATTB(00) = STATTB(00) + 1;
340   5           END;
341   4         END;
342   3         ELSE DO;
                    /* IT IS AT THIS POINT THAT OTHER CONTROL CODES WILL
                            BE INCORPORATED INTO THE NETWORK  */
343   4             DESTINATION = 8;
344   4             STATTB(02) = STATTB(02) + 1;
345   4             STATTB(00) = STATTB(00) + 1;
346   4         END;
347   3       END;
348   2       RETURN DESTINATION_ADDRESS;
349   2     END DET_DEST_ONE;
```

```
/*****************************************************************************
 PROCEDURE DET_DEST_TWO   DETERMINES THE DESTINATION OF DATA FROM LOCAL HOST

      THE PURPOSE OF THIS PROCEDURE IS TO DETERMINE THE DESTINATION OF DATA
       COMING FROM A HOST CONNECTED TO LOCAL CHANNEL-2.

      INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING THE TABLE
              LOCATION OF THE PACKET TO BE EVALUATED AND A LOCATION FOR THE
              DESTINATION TO BE PLACED..

      PROCESSING - THE PROCEDURE FIRST EXTRACTS THE CONTROL_CODE FROM THE INCOMMING
              DATA TO DETERMINE WHICH ROUTING SCHEME IS USED. THE COUNTRY_CODE
              AND NETWORK_CODE ARE THEN EXTRACTED TO DETERMINE THE DATA'S
              DESTINATION. IF THE DATA IS DESTINED FOR THE NETWORK SIDE OF THE
              UNID THEN THE HOST_CODE IS ALSO EXTRACTED SO THAT THE
              DESTINATION_ADDRESS CAN BE DETERMINED.

      OUTPUT - THIS PROCEDURE PLACES THE DESTINATION (6 OR 5) IN MEMORY
              FOR THE PASSING ROUTINE AND IT RETURNS
              THE DESTINATION_ADDRESS. AN EXAMPLE OF A DESTINATION_ADDRESS
              IS 21, WHICH INDICATES UNID=2 AND CHANNEL=1.
```

```
                 INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE ROUTE_IN.

                 NOTES: 1. BYTE AND OFH WILL MASK OUT THE UPPER 4-BITS.
                        2. BYTE AND OFOH WILL MASK OUT THE LOWER 4-BITS.
                        3. BYTE / 01H WILL RETURN A VALUE 0-15 FROM THE LOWER 4-BITS.
                        4. BYTE / 10H WILL RETURN A VALUE 0-15 FROM THE UPPER 4-BITS.

           *************************************************************************/

350  1     DET_DEST_TWO: PROCEDURE BYTE REENTRANT;

351  2       DECLARE BITS BYTE,
                     NEWBITS BYTE,
                     LASTBITS BYTE,
                     CONTROL_CODE BYTE,
                     COUNTRY_CODE BYTE,
                     NETWORK_CODE BYTE,
                     HOST_CODE WORD,
                     DESTINATION_ADDRESS BYTE;

352  2         HOST_CODE = 0;
353  2         CONTROL_CODE = LCO2TB(LCO2NS + 16) AND OFOH;
354  2         IF CONTROL_CODE = 00 THEN DO;
356  3           COUNTRY_CODE = LCO2TB(LCO2NS + 16) AND OFH;
357  3           IF COUNTRY_CODE <= MAX_COUNTRY_CODE THEN DO;
359  4             NETWORK_CODE = LCO2TB(LCO2NS + 17) AND OFOH;
360  4             IF (NETWORK_CODE /10H) <= MAX_NETWORK_CODE THEN DO;
362  5               IF (COUNTRY_CODE <> THIS_COUNTRY_CODE) OR ((NETWORK_CODE /10H)<>
                     THIS_UNID_NBR) THEN DO;
364  6                 DESTINATION = 6;
365  6                 BITS = ROL(LCO2TB(LCO2NS + 17),4);
366  6                 NEWBITS = BITS AND OFOH;
367  6                 BITS = ROR(LCO2TB(LCO2NS + 18),4);
368  6                 LASTBITS = BITS AND OFH;
369  6                 HOST_CODE = DOUBLE(BITS OR LASTBITS);
370  6                 IF (HOST_CODE >=0) AND (HOST_CODE <= 63) THEN
371  6                       DESTINATION_ADDRESS = NETWORK_CODE OR 01H;
372  6                 ELSE IF (HOST_CODE >=64) AND (HOST_CODE <= 127) THEN
373  6                       DESTINATION_ADDRESS = NETWORK_CODE OR 02H;
374  6                 ELSE IF (HOST_CODE >= 128) AND (HOST_CODE <= 191) THEN
375  6                       DESTINATION_ADDRESS = NETWORK_CODE OR 03H;
376  6                 ELSE IF (HOST_CODE >= 192) AND (HOST_CODE <=255) THEN
377  6                       DESTINATION_ADDRESS = NETWORK_CODE OR 04H;
378  6                 ELSE DO;
379  7                       DESTINATION = 5;
380  7                 END;
381  6               END;
382  5               ELSE DO;
383  6                     DESTINATION = 8;
384  6                     STATTB(04) = STATTB(04) + 1;
385  6                     STATTB(00) = STATTB(00) + 1;
386  6               END;
387  5             END;
388  4             ELSE DO;
389  5                 DESTINATION = 8;
390  5                 STATTB(03) = STATTB(03) + 1;
391  5                 STATTB(00) = STATTB(00) + 1;
```

```
392   5              END;
393   4            END;
394   3            ELSE DO;
                        /* IT IS AT THIS POINT THAT OTHER CONTROL CODES WILL
                               BE INCORPORATED INTO THE NETWORK  */
395   4              DESTINATION = 8;
396   4              STATTB(02) = STATTB(02) + 1;
397   4              STATTB(00) = STATTB(00) + 1;
398   4            END;
399   3          END;
400   2          RETURN DESTINATION_ADDRESS;
401   2        END DET_DEST_TWO;

      /**************************************************************************
       PROCEDURE DET_DEST_THREE   DETERMINES THE DESTINATION OF DATA FROM LOCAL HOST

            THE PURPOSE OFTHIS PROCEDURE IS TO DETERMINE THE DESTINATION OF DATA
             COMING FROM A HOST CONNECTED TO LOCAL CHANNEL-3.

            INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING THE TABLE
                    LOCATION OF THE PACKET TO BE EVALUATED AND A LOCATION FOR THE
                    DESTINATION TO BE PLACED..

            PROCESSING - THE PROCEDURE FIRST EXTRACTS THE CONTROL_CODE FROM THE INCOMMING
                    DATA TO DETERMINE WHICH ROUTING SCHEME IS USED. THE COUNTRY_CODE
                    AND NETWORK_CODE ARE THEN EXTRACTED TO DETERMINE THE DATA'S
                    DESTINATION. IF THE DATA IS DESTINED FOR THE NETWORK SIDE OF THE
                    UNID THEN THE HOST_CODE IS ALSO EXTRACTED SO THAT THE
                    DESTINATION_ADDRESS CAN BE DETERMINED.

            OUTPUT - THIS PROCEDURE PLACES THE DESTINATION (6 OR 5) IN MEMORY
                    FOR THE PASSING ROUTINE AND IT RETURNS
                    THE DESTINATION_ADDRESS. AN EXAMPLE OF A DESTINATION_ADDRESS
                    IS 21, WHICH INDICATES UNID=2 AND CHANNEL=1.

            INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE ROUTE_IN.

            NOTES: 1. BYTE AND OFH WILL MASK OUT THE UPPER 4-BITS.
                   2. BYTE AND OFOH WILL MASK OUT THE LOWER 4-BITS.
                   3. BYTE / 01H WILL RETURN A VALUE 0-15 FROM THE LOWER 4-BITS.
                   4. BYTE / 10H WILL RETURN A VALUE 0-15 FROM THE UPPER 4-BITS.

      **************************************************************************/

402   1      DET_DEST_THREE: PROCEDURE BYTE REENTRANT;

403   2        DECLARE BITS BYTE,
                       NEWBITS BYTE,
                       LASTBITS BYTE,
                       CONTROL_CODE BYTE,
                       COUNTRY_CODE BYTE,
                       NETWORK_CODE BYTE,
                       HOST_CODE WORD,
                       DESTINATION_ADDRESS BYTE;

404   2          HOST_CODE = 0;
```

```
405  2          CONTROL_CODE = LCO3TB(LCO3NS + 16) AND OFOH;
406  2          IF CONTROL_CODE = 00 THEN DO;
408  3            COUNTRY_CODE = LCO3TB(LCO3NS + 16) AND OFH;
409  3            IF COUNTRY_CODE <= MAX_COUNTRY_CODE THEN DO;
411  4              NETWORK_CODE = LCO3TB(LCO3NS + 17) AND OFOH;
412  4              IF (NETWORK_CODE /10H) <= MAX_NETWORK_CODE THEN DO;
414  5                IF (COUNTRY_CODE <> THIS_COUNTRY_CODE) OR ((NETWORK_CODE /10H)<>
                      THIS_UNID_NBR) THEN DO;
416  6                  DESTINATION = 6;
417  6                  BITS = ROL(LCO3TB(LCO3NS + 17),4);
418  6                  NEWBITS = BITS AND OFOH;
419  6                  BITS = ROR(LCO3TB(LCO3NS + 18),4);
420  6                  LASTBITS = BITS AND OFH;
421  6                  HOST_CODE = DOUBLE(BITS OR LASTBITS);
422  6                  IF (HOST_CODE >=0) AND (HOST_CODE <= 63) THEN
423  6                        DESTINATION_ADDRESS = NETWORK_CODE OR 01H;
424  6                  ELSE IF (HOST_CODE >=64) AND (HOST_CODE <= 127) THEN
425  6                        DESTINATION_ADDRESS = NETWORK_CODE OR 02H;
426  6                  ELSE IF (HOST_CODE >= 128) AND (HOST_CODE <= 191) THEN
427  6                        DESTINATION_ADDRESS = NETWORK_CODE OR 03H;
428  6                  ELSE IF (HOST_CODE >= 192) AND (HOST_CODE <=255) THEN
429  6                        DESTINATION_ADDRESS = NETWORK_CODE OR 04H;
                    END;
431  5              ELSE DO;
432  6                        DESTINATION = 5;
433  6                  END;
434  5                END;
435  4                ELSE DO;
436  5                      DESTINATION = 8;
437  5                      STATTB(04) = STATTB(04) + 1;
438  5                      STATTB(00) = STATTB(00) + 1;
439  5                END;
440  4              END;
441  3              ELSE DO;
442  4                  DESTINATION = 8;
443  4                  STATTB(03) = STATTB(03) + 1;
444  4                  STATTB(00) = STATTB(00) + 1;
445  4              END;
446  3            END;
447  2            ELSE DO;
                        /* IT IS AT THIS POINT THAT OTHER CONTROL CODES WILL
                                BE INCORPORATED INTO THE NETWORK  */
448  3                DESTINATION = 8;
449  3                STATTB(02) = STATTB(02) + 1;
450  3                STATTB(00) = STATTB(00) + 1;
451  3            END;
452  2          RETURN DESTINATION_ADDRESS;
453  2        END DET_DEST_THREE;


               /***************************************************************************
               PROCEDURE DET_DEST_FOUR   DETERMINES THE DESTINATION OF DATA FROM LOCAL HOST

                    THE PURPOSE OFTHIS PROCEDURE IS TO DETERMINE THE DESTINATION OF DATA
                     COMING FROM A HOST CONNECTED TO LOCAL CHANNEL-4.

                    INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING THE TABLE
                            LOCATION OF THE PACKET TO BE EVALUATED AND A LOCATION FOR THE
```

```
                    DESTINATION TO BE PLACED..

                PROCESSING - THE PROCEDURE FIRST EXTRACTS THE CONTROL_CODE FROM THE INCOMMING
                    DATA TO DETERMINE WHICH ROUTING SCHEME IS USED. THE COUNTRY_CODE
                    AND NETWORK_CODE ARE THEN EXTRACTED TO DETERMINE THE DATA'S
                    DESTINATION. IF THE DATA IS DESTINED FOR THE NETWORK SIDE OF THE
                    UNID THEN THE HOST_CODE IS ALSO EXTRACTED SO THAT THE
                    DESTINATION_ADDRESS CAN BE DETERMINED.

                OUTPUT - THIS PROCEDURE PLACES THE DESTINATION (6 OR 5) IN MEMORY
                    FOR THE PASSING ROUTINE AND IT RETURNS
                    THE DESTINATION_ADDRESS. AN EXAMPLE OF A DESTINATION_ADDRESS
                    IS 21, WHICH INDICATES UNID=2 AND CHANNEL=1.

                INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE ROUTE_IN.

                NOTES: 1. BYTE AND 0FH WILL MASK OUT THE UPPER 4-BITS.
                       2. BYTE AND 0F0H WILL MASK OUT THE LOWER 4-BITS.
                       3. BYTE / 01H WILL RETURN A VALUE 0-15 FROM THE LOWER 4-BITS.
                       4. BYTE / 10H WILL RETURN A VALUE 0-15 FROM THE UPPER 4-BITS.

            ************************************************************************/

 454   1    DET_DEST_FOUR: PROCEDURE BYTE REENTRANT;

 455   2      DECLARE BITS BYTE,
                      NEWBITS BYTE,
                      LASTBITS BYTE,
                      CONTROL_CODE BYTE,
                      COUNTRY_CODE BYTE,
                      NETWORK_CODE BYTE,
                      HOST_CODE WORD,
                      DESTINATION_ADDRESS BYTE;

 456   2        HOST_CODE = 0;
 457   2        CONTROL_CODE = LC04TB(LC04NS + 16) AND 0F0H;
 458   2        IF CONTROL_CODE = 00 THEN DO;
 460   3          COUNTRY_CODE = LC04TB(LC04NS + 16) AND 0FH;
 461   3          IF COUNTRY_CODE <= MAX_COUNTRY_CODE THEN DO;
 463   4            NETWORK_CODE = LC04TB(LC04NS + 17) AND 0F0H;
 464   4            IF (NETWORK_CODE /10H) <= MAX_NETWORK_CODE THEN DO;
 466   5              IF (COUNTRY_CODE <> THIS_COUNTRY_CODE) OR ((NETWORK_CODE /10H)<>
                      THIS_UNID_NBR) THEN DO;
 468   6                DESTINATION = 6;
 469   6                BITS = ROL(LC04TB(LC04NS + 17),4);
 470   6                NEWBITS = BITS AND 0F0H;
 471   6                BITS = ROR(LC04TB(LC04NS + 18),4);
 472   6                LASTBITS = BITS AND 0FH;
 473   6                HOST_CODE = DOUBLE(BITS OR LASTBITS);
 474   6                IF (HOST_CODE >=0) AND (HOST_CODE <= 63) THEN
 475   6                      DESTINATION_ADDRESS = NETWORK_CODE OR 01H;
 476   6                ELSE IF (HOST_CODE >=64) AND (HOST_CODE <= 127) THEN
 477   6                      DESTINATION_ADDRESS = NETWORK_CODE OR 02H;
 478   6                ELSE IF (HOST_CODE >= 128) AND (HOST_CODE <= 191) THEN
 479   6                      DESTINATION_ADDRESS = NETWORK_CODE OR 03H;
 480   6                ELSE IF (HOST_CODE >= 192) AND (HOST_CODE <=255) THEN
 481   6                      DESTINATION_ADDRESS = NETWORK_CODE OR 04H;
 482   6                ELSE DO;
```

```
483   7                         DESTINATION = 5;
484   7                   END;
485   6                 END;
486   5                 ELSE DO;
487   6                       DESTINATION = 8;
488   6                       STATTB(04) = STATTB(04) + 1;
489   6                       STATTB(00) = STATTB(00) + 1;
490   6                 END;
491   5               END;
492   4               ELSE DO;
493   5                   DESTINATION = 8;
494   5                   STATTB(03) = STATTB(03) + 1;
495   5                   STATTB(00) = STATTB(00) + 1;
496   5               END;
497   4             END;
498   3             ELSE DO;
                        /* IT IS AT THIS POINT THAT OTHER CONTROL CODES WILL
                                BE INCORPORATED INTO THE NETWORK  */
499   4               DESTINATION = 8;
500   4               STATTB(02) = STATTB(02) + 1;
501   4               STATTB(00) = STATTB(00) + 1;
502   4             END;
503   3           END;
504   2           RETURN DESTINATION_ADDRESS;
505   2         END DET_DEST_FOUR;

             /***********************************************************************/

             /************************************************************************
              PROCEDURE DET_DEST_LL   DETERMINES THE DESTINATION OF DATA TO A LOCAL HOST

                   THE PURPOSE OF THIS PROCEDURE IS TO DETERMINE  THE DESTINATION OF DATA
                RESIDING IN THE LOCAL TABLE.

                INPUT - THE INPUT IS TWO ASCII CHARACTERS INDICATING THE TABLE LOCATION
                        OF THE DATA TO BE EVALUATED.

                PROCESSING - THE PROCEDURE FIRST EXTRACTS THE CONTROL_CODE FROM THE DATA
                        TO DETERMINE WHICH ROUTING SCHEME IS USED. THEN THE HOST_CODE IS
                        EXTRACTED TO DETERMINE WHICH LOCAL CHANNEL THE DATA SHOULD BE
                        TRANSMITTED OUT OF.

                OUTPUT - THIS PROCEDURE RETURNS THE DESTINATION ('01','02','03', OR '04')
                        TO WHICH THE DATA SHOULD BE TRANSFERED.

                INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE ROUTE_OUT.

                NOTES - 1. BYTE AND 0FH WILL MASK OUT THE UPPER 4-BITS.
                        2. BYTE AND 0F0H WILL MASK OUT THE LOWER 4-BITS.
                        3. BYTE / 01H WILL RETURN A VALUE 0-15 FROM THE LOWER 4-BITS.
                        4. BYTE / 10H WILL RETURN A VALUE 0-15 FROM THE UPPER 4-BITS.

             ***********************************************************************/
506   1      DET_DEST_LL: PROCEDURE INTEGER REENTRANT;
507   2          DECLARE   CONTROL_CODE  BYTE,
                           DESTINATION   INTEGER,
                              NEWBITS    BYTE,
```

```
                                    HOST_CODE  WORD,
                                         BITS  BYTE,
                                    LASTBITS   BYTE;

 508   2          BITS = 0;
 509   2          NEWBITS = 0;
 510   2          CONTROL_CODE = LCLCTB(LCLCNS + 16) AND OFOH;
 511   2          IF CONTROL_CODE = 00 THEN DO;
 513   3            BITS = ROL(LCLCTB(LCLCNS + 17),4);
 514   3            NEWBITS = BITS AND OFOH;
 515   3            BITS = ROR(LC01TB(LC01NS + 18),4);
 516   3            LASTBITS = BITS AND OFH;
 517   3            HOST_CODE = DOUBLE(BITS OR LASTBITS);
 518   3            IF (HOST_CODE >= 0 AND HOST_CODE <= 63) THEN
 519   3              DESTINATION = 1;
 520   3            ELSE IF (HOST_CODE >= 64 AND HOST_CODE <= 127) THEN
 521   3              DESTINATION = 2;
 522   3            ELSE IF (HOST_CODE >= 128 AND HOST_CODE <= 191) THEN
 523   3              DESTINATION = 3;
 524   3            ELSE IF (HOST_CODE >= 192 AND HOST_CODE <= 255) THEN
 525   3              DESTINATION = 4;
 526   3            ELSE DO;
 527   4              DESTINATION = 8;
 528   4              STATTB(02) = STATTB(02) + 1; /* INCREMENT CONTROL_CODE ERROR */
 529   4              STATTB(01) = STATTB(01) + 1; /* INCREMENT LOCAL ERROR COUNT */
 530   4            END;
 531   3          END;
 532   2          ELSE DO;
 533   3              DESTINATION = 8;
 534   3              STATTB(02) = STATTB(02) + 1;
 535   3              STATTB(01) = STATTB(01) + 1;
 536   3          END;
 537   2        RETURN  DESTINATION;
 538   2      END DET_DEST_LL;

        /**********************************************************************/

        /**********************************************************************
        PROCEDURE DET_DEST_NL    DETERMINES THE DESTINATION OF DATA FROM THE NETWORK

               THE PURPOSE OF THIS PROCEDURE IS TO DETERMINE THE DESTINATION OF DATA
            COMING FROM THE NETWORK SIDE OF THE UNID TO A LOCAL HOST.

            INPUT - THE INPUT IS AN INTEGER VALUE INDICATING THE TABLE LOCATION OF
                    THE DATA TO BE EVALUATED.

            PROCESSING - THE PROCEDURE EXTRACTS THE DESTINATION_ADDRESS FROM THE
                    SECOND BYTE OF THE DATA PACKET AND RETURNS THE CORRESPONDING
                    DESTINATION.

            OUTPUT - THIS PROCEDURE RETURNS THE DESTINATION (1,2,3, OR 4) OF THE
                    DATA COMING FROM THE NETWORK TO A LOCAL HOST.

            INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE ROUTE_IN.

            NOTES: 1. BYTE AND 07H WILL MASK OUT THE UPPER 5-BITS.
```

```
                 ***********************************************************************/

 539   1         DET_DEST_NL: PROCEDURE(TABLE) INTEGER REENTRANT;
 540   2             DECLARE  TABLE  WORD;

 541   2             DECLARE  DESTINATION  INTEGER,
                                    BITS  WORD;

 542   2              BITS = 0;
 543   2              BITS = DOUBLE((NTLCTB(NTLCNS + 0) AND 07H));
 544   2              IF (BITS <= 4 AND BITS >= 1) THEN
 545   2                DO CASE BITS;
 546   3                  DESTINATION = 1;
 547   3                  DESTINATION = 2;
 548   3                  DESTINATION = 3;
 549   3                  DESTINATION = 4;
 550   3                END;
 551   2              ELSE DO;
 552   3                DESTINATION = 8;
 553   3                STATTB(01) = STATTB(01) + 1;  /*  INCREMENT LOCAL ERROR COUNT  */
 554   3              END;
 555   2           RETURN DESTINATION;
 556   2         END DET_DEST_NL;

                 /*********************************************************************/

                 /*********************************************************************
                 PROCEDURE  LD_TAB_HSKP     LOAD TABLE HOUSEKEEP

                           THE PURPOSE OF THIS PROCEDURE IS TO HOUSEKEEP A SPECIFIED BUFFER
                       TABLE AFTER LOADING OF THE USER DATA FROM THE HOST.

                       INPUT - THE INPUT IS AN INTEGER INDICATING THE TABLE REQUIRING
                               CHANGES.

                       PROCESSING - THE PROCEDURE DETERMINES THE TABLE TO BE PROCESSED,
                               ADVANCES THE NEXT_EMPTY_BYTE POINTER BY ONE PACKET_SIZE,
                               AND ADJUSTS FOR BUFFER WRAP IF NECESSARY.

                       OUTPUT - THE SPECIFIED TABLE HAS ITS NEXT_EMPTY_BYTE POINTER
                               ADVANCED BY THE LENGTH OF A SINGLE PACKET.

                       INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE ROUTE_IN AND ROUTE_OUT.

                 ************************************************************************/
 557   1         LD_TAB_HSKP:PROCEDURE(TABLE) REENTRANT;
 558   2                 DECLARE  TABLE WORD;

 559   2               IF (TABLE >=1 AND TABLE <= 6) THEN
 560   2                 DO CASE TABLE;
 561   3                   ;                      /* CASE ZERO IS NULL */
 562   3                   DO;                     /* START CASE ONE  */
 563   4                     LC01NE = LC01NE + DATA_SIZE; /* ADVANCE NEXT_EMPTY POINTER */
 564   4                     IF LC01NE >= LC01SZ THEN
 565   4                        LC01NE = 0;
 566   4                   END;
```

```
567   3              DO;                    /* START CASE TWO  */
568   4                LCO2NE = LCO2NE + DATA_SIZE;  /* ADVANCE NEXT_EMPTY POINTER  */
569   4                IF LCO2NE >= LCO2SZ THEN
570   4                   LCO2NE = 0;
571   4              END;

572   3              DO;
573   4                LCO3NE = LCO3NE + DATA_SIZE;
574   4                IF LCO3NE >= LCO3SZ THEN
575   4                   LCO3NE = 0;
576   4              END;

577   3              DO;
578   4                LCO4NE = LCO4NE + DATA_SIZE;
579   4                IF LCO4NE >= LCO4SZ THEN
580   4                   LCO4NE = 0;
581   4              END;

582   3              DO;                /* START OF CASE FIVE  */
583   4                LCLCNE = LCLCNE + DATA_SIZE;
584   4                IF LCLCNE >= LCLCSZ THEN
585   4                   LCLCNE = 0;
586   4              END;

587   3              DO;                /* START OF CASE SIX  */
588   4                LCNTNE = LCNTNE + DATA_SIZE;
589   4                IF LCNTNE >= LCNTSZ THEN
590   4                   LCNTNE = 0;
591   4              END;
592   3           END;

593   2           ELSE HSKP_ERR = HSKP_ERR + 1;

594   2      END LD_TAB_HSKP;

        /***************************************************************************/

        /***************************************************************************
         PROCEDURE   SRVC_TAB_HSKP       SERVICE TABLE HOUSEKEEP

                    THE PURPOSE OF THIS PROCEDURE IS TO HOUSEKEEP A SPECIFIED
                  BUFFER TABLE AFTER SERVICING (REMOVING A PACKET).

            INPUT - THE INPUT IS AN INTEGER VALUE INDICATING THE TABLE THAT
                    REQUIRES HOUSEKEEPING.

            PROCESSING - THE PROCEDURE DETERMINES THE TABLE TO BE PROCESSED,
                    ADVANCES THE NEXT_BYTE_TO_BE_SERVICED POINTER BY A PACKET_SIZE,
                    AND ADJUSTS FOR BUFFER WRAP IF NECESSARY.

            OUTPUT - THE SPECIFIED TABLE HAS ITS NEXT_BYTE_TO_BE_SERVICED POINTER
                    ADVANCED BY THE LENGTH OF A SINGLE PACKET.

            INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE ROUTE_IN AND ROUTE_OUT.

        ***************************************************************************/
```

```
595   1       SRVC_TAB_HSKP:PROCEDURE(TAB) REENTRANT;
596   2           DECLARE  TAB  WORD;

597   2            IF (TAB >= 1 AND TAB <=7) THEN
598   2              DO CASE TAB;

599   3                ;                  /*  CASE ZERO IS NULL  */

600   3                DO;                /*  CASE ONE STARTS HERE  */
601   4                  LC01NS = LC01NS + DATA_SIZE;
602   4                  IF LC01NS >= LC01SZ THEN
603   4                     LC01NS = 0;
604   4                END;

605   3                DO;                /*  CASE TWO STARTS HERE  */
606   4                  LC02NS = LC02NS + DATA_SIZE;
607   4                  IF LC02NS >= LC02SZ THEN
608   4                     LC02NS = 0;
609   4                END;

610   3                DO;
611   4                  LC03NS = LC03NS + DATA_SIZE;
612   4                  IF LC03NS >= LC03SZ THEN
613   4                     LC03NS = 0;
614   4                END;

615   3                DO;               /*  CASE FOUR STARTS HERE  */
616   4                  LC04NS = LC04NS + DATA_SIZE;   /* ADVANCE NEXT SERVICE POINTER */
617   4                  IF LC04NS >= LC04SZ THEN
618   4                     LC04NS = 0;
619   4                END;

620   3                DO;               /*  CASE FIVE STARTS HERE  */
621   4                  LCLCNS = LCLCNS + DATA_SIZE;   /* ADVANCE NEXT SERVICE POINTER */
622   4                  IF LCLCNS >= LCLCSZ THEN
623   4                     LCLCNS = 0;
624   4                END;

625   3                ;        /*  CASE SIX IS A NULL STATEMENT  */

626   3                DO;              /*  CASE SEVEN STARTS HERE  */
627   4                  NTLCNS = NTLCNS + DATA_SIZE;   /* ADVANCE NEXT SERVICE POINTER */
628   4                  IF NTLCNS >= NTLCSZ THEN
629   4                    NTLCNS = 0;
630   4                END;
631   3              END;

632   2              ELSE  HSKP_ERR = HSKP_ERR + 1;
633   2         END SRVC_TAB_HSKP;

       /***************************************************************************/

       /***************************************************************************
       PROCEDURE   TRNMIT_PKT      TRANSMIT A PACKET

                         THE PURPOSE OF THIS PROCEDURE IS TO SET UP THE DATA PORT FOR
                     PACKET TRANSMISSION.
```

```
                 INPUT - THE USART DATA PORT ADDRESS IS INPUT TO TRNMIT_PKT.

                 PROCESSING - TWO GLOBAL VALUES, TDAADD (DATA ADDRESS) AND TPRADD
                        (PORT ADDRESS), ARE LOADED WITH THE CORRECT INITIAL VALUES.
                        THE PROCEDURE THEN SETS THE INTERRUPT FOR THE TRANSMISSION OF
                        A PACKET.

                 OUTPUT - THE INTERRUPT IS SET ON THE CHANNEL SPECIFIED BY PRTADD.

                 INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE ROUTE_OUT.

             ****************************************************************************/
634   1      TRNMIT_PKT:PROCEDURE(PRTADD);
635   2                 DECLARE  PRTADD  BYTE;
636   2         DISABLE;
637   2           OUTPUT(B$OUT) = TRANS$STATE;
638   2           OUTPUT(C$CNTRL) = PRTADD;
639   2           OUTPUT(C$CNTRL) = END$WRITE$D;
640   2         ENABLE;
641   2      END TRNMIT_PKT;

             /****************************************************************************/
             /* PROCEDURE  BUILD_I_PACKET  PROCEDURE FOR TRANSFORMING THE USER DATA      */
             /*                            INTO A DATA_PACKET FOR TRANSFER TO THE        */
             /*                            NETWORK SIDE OF THE UNID.                     */
             /*                                                                          */
             /*               THE PURPOSE OF THIS PROCEDURE IS TO TRANSFORM THE HOST'S   */
             /*          USER DATA DELIVERED TO ONE OF THE LOCAL INPUT BUFFERS INTO A    */
             /*          DATA_PACKET TO BE PLACED IN THE LOCAL TO NETWORK BUFFER. THIS   */
             /*          PROCEDURE ADDS THE FIVE HEADER BYTES, AS FOLLOWS:               */
             /*             1.  1 BYTE FOR THE DESTINATION ADDRESS.                      */
             /*             2.  1 BYTE FOR THE SOURCE ADDRESS.                           */
             /*             3.  1 BYTE FOR THE SEQUENCE NUMBER.                          */
             /*             4.  1 BYTE FOR A SPARE  (SPARE_01).                          */
             /*             5.  1 BYTE FOR A SPARE  (SPARE_02).                          */
             /*                                                                          */
             /*        INPUT - THIS PROCEDURE RECEIVES AN INTEGER THAT INDICATES THE     */
             /*                LOCAL INPUT BUFFER WHERE THE INCOMMING HOST DATA IS       */
             /*                LOCATED.                                                  */
             /*        PROCESSING - THE PROCEDURE BEGINS WITH THE PASSING OF THE TABLE   */
             /*                WHERE THE HOST'S DATA IS LOCATED. THE SOURCE AND          */
             /*                DESTINATION ADDRESS (SOURCE_ADDRESS, DESTINATION_ADDRESS) */
             /*                IS SUPPLIED BY THE DET_DEST PROCEDURE. FOR NOW, THE       */
             /*                SEQUENCE NUMBER AND BOTH SPARE BYTES ARE SET TO ZERO. IN  */
             /*                THE FUTURE, THESE BYTES WILL REFLECT X.25 PROTOCOL.       */
             /*                                                                          */
             /*        OUTPUT - THIS PROCEDURE PLACES THE FIRST FIVE BYTES INTO THE      */
             /*                LOCAL TO NETWORK BUFFER TABLE BEFORE THE PACKET IS        */
             /*                TRANSFERED OVER TO THE NETWORK SIDE OF THE UNID.          */
             /*                                                                          */
             /*        INTERFACE - THE PROCEDURE IS CALLED FROM PROCEDURE ROUTE_IN FOR   */
             /*                THOSE DATA PACKETS DESTINED FOR THE NETWORK ONLY.         */
             /*                                                                          */
             /*        NOTE:  THE HEADER INFORMATION SUPPPLIED IS FOR DATAGRAM SERVICE   */
             /*               ONLY. ALSO, THE SEQUENCE AND SPARE BYTES ARE SET TO ZERO   */
             /*               TO ALLOW THE UNIDS MINIMAL OPERATIONAL CAPABILITY. FOR     */
```

```
                 /*              FUTURE SOFTWARE ENHANCEMENTS THIS MUST BE MODIFIED.        */
                 /*                                                                         */
                 /***************************************************************************/
642   1          BUILD_I_PACKET: PROCEDURE(LIST,SOURCE_ADDRESS,DESTINATION_ADDRESS) REENTRANT;
643   2                          DECLARE  LIST  WORD, (SOURCE_ADDRESS,DESTINATION_ADDRESS) BYTE;

                               /*  DECLARATIONS FOR THE LOCAL VARIABLES   */

644   2                    DECLARE  DESTINATION_BYTE   BYTE,
                                    SOURCE_BYTE        BYTE,
                                    SEQUENCE_BYTE      BYTE,
                                    SPARE_01_BYTE      BYTE,
                                    SPARE_02_BYTE      BYTE;

645   2                 IF(LIST >= 1 AND LIST <= 4) THEN
646   2                    DO CASE LIST;

647   3                       ;            /* ZERO CASE IS NULL AND IS AN ERROR CASE  */

648   3                       DO;   /* CASE ONE FOR CHANNEL ONE  */
649   4                         DESTINATION_BYTE = DESTINATION_ADDRESS;
650   4                         SOURCE_BYTE = SOURCE_ADDRESS;
651   4                         SEQUENCE_BYTE = 0;
652   4                         SPARE_01_BYTE =0;
653   4                         SPARE_02_BYTE =0;
654   4                         LCNTTB(LCNTNE + 0) = DESTINATION_BYTE;
655   4                         LCNTTB(LCNTNE + 1) = SOURCE_BYTE;
656   4                         LCNTTB(LCNTNE + 2) = SEQUENCE_BYTE;
657   4                         LCNTTB(LCNTNE + 3) = SPARE_01_BYTE;
658   4                         LCNTTB(LCNTNE + 4) = SPARE_02_BYTE;
659   4                       END;

660   3                       DO;   /* CASE TWO FOR LOCAL CHANNEL TWO  */
661   4                         DESTINATION_BYTE = DESTINATION_ADDRESS;
662   4                         SOURCE_BYTE = SOURCE_ADDRESS;
663   4                         SEQUENCE_BYTE = 0;
664   4                         SPARE_01_BYTE =0;
665   4                         SPARE_02_BYTE =0;
666   4                         LCNTTB(LCNTNE + 0) = DESTINATION_BYTE;
667   4                         LCNTTB(LCNTNE + 1) = SOURCE_BYTE;
668   4                         LCNTTB(LCNTNE + 2) = SEQUENCE_BYTE;
669   4                         LCNTTB(LCNTNE + 3) = SPARE_01_BYTE;
670   4                         LCNTTB(LCNTNE + 4) = SPARE_02_BYTE;
671   4                       END;

672   3                       DO;    /*  CASE THREE FOR LOCAL CHANNEL THREE    */
673   4                         DESTINATION_BYTE = DESTINATION_ADDRESS;
674   4                         SOURCE_BYTE = SOURCE_ADDRESS;
675   4                         SEQUENCE_BYTE = 0;
676   4                         SPARE_01_BYTE =0;
677   4                         SPARE_02_BYTE =0;
678   4                         LCNTTB(LCNTNE + 0) = DESTINATION_BYTE;
679   4                         LCNTTB(LCNTNE + 1) = SOURCE_BYTE;
680   4                         LCNTTB(LCNTNE + 2) = SEQUENCE_BYTE;
681   4                         LCNTTB(LCNTNE + 3) = SPARE_01_BYTE;
682   4                         LCNTTB(LCNTNE + 4) = SPARE_02_BYTE;
683   4                       END;
```

```
684   3              DO;    /*  CASE FOUR FOR LOCAL CHANNEL FOUR   */
685   4                DESTINATION_BYTE = DESTINATION_ADDRESS;
686   4                SOURCE_BYTE = SOURCE_ADDRESS;
687   4                SEQUENCE_BYTE = 0;
688   4                SPARE_01_BYTE =0;
689   4                SPARE_02_BYTE =0;
690   4                LCNTTB(LCNTNE + 0) = DESTINATION_BYTE;
691   4                LCNTTB(LCNTNE + 1) = SOURCE_BYTE;
692   4                LCNTTB(LCNTNE + 2) = SEQUENCE_BYTE;
693   4                LCNTTB(LCNTNE + 3) = SPARE_01_BYTE;
694   4                LCNTTB(LCNTNE + 4) = SPARE_02_BYTE;
695   4              END;
696   3           END;

697   2           ELSE DO;
698   3             STATTB(0) = STATTB(0) + 1;
699   3           END;

700   2        END BUILD_I_PACKET;

           /*************************************************************************/
           /* PROCEDURE   ROUTE_IN    ROUTES IN EITHER DATA_PACKETS OR HOST DATA      */
           /*                                                                        */
           /*              THE PURPOSE OF THIS PROCEDURE IS TO EITHER ROUTE HOST DATA */
           /*         FROM THE FOUR LOCAL INPUT BUFFERS TO THEIR CORRECT OUTPUT       */
           /*         BUFFER OR ROUTE DATA_PACKETS FROM THE NETWORK SIDE OF THE UNID  */
           /*         TO THEIR CORRECT OUTPUT LOCAL BUFFER MINUS THE DATA_PACKET      */
           /*         HEADER.                                                         */
           /*                                                                        */
           /*       INPUT - HOST DATA OR DATA PACKETS ARE ROUTED VIA EVALUATION OF    */
           /*             LCXXTB POINTERS AND INTERNAL PACKET ROUTING INFORMATION.    */
           /*       PROCESSENG - THE PROCEDURE CHECKS EACH OF THE LOCAL INPUT BUFFER'S */
           /*              POINTERS FOR HOST DATA ARRIVAL. IF ANY HOST DATA IS READY, */
           /*              THE DESTINATION IS DETERMINED VIA PROCEDURE DET_DEST. ONCE */
           /*              THE DESTINATION IS DETERMINED THEN THE DATA IS EITHER SENT */
           /*              TO A LOCAL BUFFER (LOCAL TO LOCAL TRANSFER) VIA PROCEDURE  */
           /*              MOVSEQ OR THE HOST DATA IS CONVERTED INTO A DATA_PACKET    */
           /*              (BUILD_I_PACKET) THEN SENT TO THE NETWORK SIDE OF THE UNID */
           /*              VIA MOVSEQ. BOTH THE BUFFER TABLE THAT IS LOADED AND THE   */
           /*              TABLE THAT IS SERVICED HAVE THEIR POINTERS HOUSECLEANED    */
           /*              BY LD_TAB_HSKP AND SRVC_TAB_HSKP.                          */
           /*        OUTPUT - EITHER A DATA_PACKET OR HOST DATA IS MOVED TO AN        */
           /*              APPROPRIATE DESTINATION BUFFER.                            */
           /*        INTERFACE - THIS PROCEDURE IS CALLED IN AN ENDLESS LOOP BY THE   */
           /*              PROCEDURE L.MAIN_U1.                                       */
           /*                                                                        */
           /*************************************************************************/
701   1        ROUTE_IN:PROCEDURE;

702   2          DECLARE  SOURCE_ADDRESS  BYTE,
                          DESTINATION_ADDRESS  BYTE;

703   2          SOURCE_ADDRESS = 0;
704   2          DESTINATION_ADDRESS = 0;
705   2          IF(((LC01NE - LC01NS) >= DATA_SIZE) OR (LC01NS > LC01NE)) THEN DO;
707   3            CALL SNDSEQ(@TP_6, SIZE(TP_6));
```

```
708   3              DESTINATION_ADDRESS = DET_DEST_ONE;
709   3              IF DESTINATION = 5 THEN DO;
711   4                  CALL SNDSEQ(@TP_7, SIZE(TP_7));
712   4                  CALL MOVB(@LC01TB(LC01NS),@LCLCTB(LCLCNE),DATA_SIZE);
713   4                  CALL LD_TAB_HSKP(5);
714   4               END;
715   3               ELSE IF DESTINATION = 6 THEN DO;
717   4                  CALL SNDSEQ(@TP_8, SIZE(TP_8));
718   4                  SOURCE_ADDRESS = 11H;
719   4                  CALL BUILD_I_PACKET(1, SOURCE_ADDRESS, DESTINATION_ADDRESS);
720   4                  CALL MOVB(@LC01TB(LC01NS),@LCNTTB(LCNTNE+5),DATA_SIZE);
721   4                  CALL LD_TAB_HSKP(6);
722   4               END;
723   3               ELSE DO;
724   4                  CALL SNDSEQ(@TP_9, SIZE(TP_9));
725   4                  STATTB(L_RI_DEST_ERR) = STATTB(L_RI_DEST_ERR) + 1;
726   4               END;
727   3               CALL LD_TAB_HSKP(1);
728   3           END;

729   2          IF(((LC02NE - LC02NS) >= DATA_SIZE) OR (LC02NS > LC02NE)) THEN DO;
731   3              CALL SNDSEQ(@TP_10, SIZE(TP_10));
732   3              DESTINATION_ADDRESS = DET_DEST_TWO;
733   3              IF DESTINATION = 5 THEN DO;
735   4                  CALL SNDSEQ(@TP_11, SIZE(TP_11));
736   4                  CALL MOVB(@LC02TB(LC02NS),@LCLCTB(LCLCNE),DATA_SIZE);
737   4                  CALL LD_TAB_HSKP(5);
738   4               END;
739   3               ELSE IF DESTINATION = 6 THEN DO;
741   4                  CALL SNDSEQ(@TP_12, SIZE(TP_12));
742   4                  SOURCE_ADDRESS = 11H;
743   4                  CALL BUILD_I_PACKET(2, SOURCE_ADDRESS, DESTINATION_ADDRESS);
744   4                  CALL MOVB(@LC02TB(LC02NS),@LCNTTB(LCNTNE+5),DATA_SIZE);
745   4                  CALL LD_TAB_HSKP(6);
746   4               END;
747   3               ELSE DO;
748   4                  CALL SNDSEQ(@TP_13, SIZE(TP_13));
749   4                  STATTB(L_RI_DEST_ERR) = STATTB(L_RI_DEST_ERR) + 1;
750   4               END;
751   3               CALL SRVC_TAB_HSKP(2);
752   3           END;

753   2          IF(((LC03NE - LC03NS) >= DATA_SIZE) OR (LC03NS > LC03NE)) THEN DO;
755   3              CALL SNDSEQ(@TP_14, SIZE(TP_14));
756   3              DESTINATION_ADDRESS = DET_DEST_THREE;
757   3              IF DESTINATION = 5 THEN DO;
759   4                  CALL SNDSEQ(@TP_15, SIZE(TP_15));
760   4                  CALL MOVB(@LC03TB(LC03NS),@LCLCTB(LCLCNE),DATA_SIZE);
761   4                  CALL LD_TAB_HSKP(5);
762   4               END;
763   3               ELSE IF DESTINATION = 6 THEN DO;
765   4                  CALL SNDSEQ(@TP_16, SIZE(TP_16));
766   4                  SOURCE_ADDRESS = 11H;
767   4                  CALL BUILD_I_PACKET(3, SOURCE_ADDRESS, DESTINATION_ADDRESS);
768   4                  CALL MOVB(@LC03TB(LC03NS),@LCNTTB(LCNTNE+5), DATA_SIZE);
769   4                  CALL LD_TAB_HSKP(6);
770   4               END;
```

```
771   3             ELSE DO;
772   4                CALL SNDSEQ(@TP_17, SIZE(TP_17));
773   4                STATTB(L_RI_DEST_ERR) = STATTB(L_RI_DEST_ERR) + 1;
774   4             END;
775   3             CALL SRVC_TAB_HSKP(3);
776   3          END;

777   2         IF(((LCO4NE - LCO4NS) >= DATA_SIZE) OR (LCO4NS > LCO4NE)) THEN DO;
779   3            CALL SNDSEQ(@TP_18, SIZE(TP_18));
780   3            DESTINATION_ADDRESS = DET_DEST_FOUR;
781   3            IF DESTINATION = 5 THEN DO;
783   4                CALL SNDSEQ(@TP_19, SIZE(TP_19));
784   4                CALL MOVB(@LCO4TB(LCO4NS),@LCLCTB(LCLCNE),DATA_SIZE);
785   4                CALL LD_TAB_HSKP(5);
786   4             END;
787   3             ELSE IF DESTINATION = 6 THEN DO;
789   4                CALL SNDSEQ(@TP_20, SIZE(TP_20));
790   4                SOURCE_ADDRESS = 11H;
791   4                CALL BUILD_I_PACKET(4, SOURCE_ADDRESS, DESTINATION_ADDRESS);
792   4                CALL MOVB(@LCO4TB(LCO4NS),@LCNTTB(LCNTNE+5), DATA_SIZE);
793   4                CALL LD_TAB_HSKP(6);
794   4             END;
795   3             ELSE DO;
796   4                CALL SNDSEQ(@TP_21, SIZE(TP_21));
797   4                STATTB(L_RI_DEST_ERR) = STATTB(L_RI_DEST_ERR) + 1;
798   4             END;
799   3             CALL SRVC_TAB_HSKP(4);
800   3          END;
801   2       END ROUTE_IN;

          /***************************************************************************/
          /*                                                                         */
          /* PROCEDURE  ROUTE_OUT   ROUTE OUT EITHER A DATA_PACKET OR HOST DATA      */
          /*                                                                         */
          /*            THE PURPOSE OF THIS PROCEDURE IS TO ROUTE PACKETS FROM THE   */
          /*        LOCAL TO LOCAL AND NETWORK TO LOCAL TABLES TO THE CORRECT        */
          /*        OUTPUT CHANNEL.                                                  */
          /*                                                                         */
          /*        INPUT - DATA PACKETS ARE ROUTED VIA EVALUATION OF LCLCTB AND     */
          /*                NTLCTB POINTERS AND INTERNAL PACKET ROUTING INFORMATION  */
          /*        PROCESSING - THE PROCEDURE CHECKS EACH INPUT BUFFER'S POINTERS   */
          /*                FOR PACKET ARRIVAL. IF A PACKET IS READY,THE DESTINATION */
          /*                IS DETERMINED VIA PROCEDURE DET_DEST, AND TRANSMITTED    */
          /*                BY SETTING THE TRANSMIT INTERRUPTS THROUGH PROCEDURE     */
          /*                TRNMIT_PKT. THE TABLE THAT WAS SERVICED (PACKET REMOVED) */
          /*                HAS ITS POINTERS UPDATED BY SRVC_TAB_HSKP. WHEN THE      */
          /*                FRAME IS TRANSMITTED TO THE HOSTS FROM THE NET TO LOCAL  */
          /*                BUFFER TABLE, THE FIRST TWO BYTES OF HEADER INFORMATION  */
          /*                ARE STRIPPED OFF BEFORE TRANSMISSION.                    */
          /*        OUTPUT - A PACKET OF DATA IS PREPARED FOR TRANSMISSION AND THE   */
          /*                INTERRUPTS ARE SET.                                      */
          /*        INTERFACE - THIS PROCEDURE IS CALLED IN AN INFINITE LOOP BY THE  */
          /*                MAIN PROGRAM.                                            */
          /*                                                                         */
          /***************************************************************************/
802   1       ROUTE_OUT: PROCEDURE;
```

```
803  2          IF((LCLCNE - LCLCNS) >= DATA_SIZE) OR (LCLCNS > LCLCNE) THEN DO;
805  3              IF(DESTINATION >= 1 AND DESTINATION <= 4) THEN
806  3                 DO CASE DESTINATION;

807  4                    ;     /*  CASE ZERO IS NULL   */

808  4                    DO;    /*  CASE ONE  */
809  5                      IF TRANS$1$RDY = TRUE THEN DO;
811  6                         CALL SNDSEQ(@TP_24, SIZE(TP_24));
812  6                         CALL MOVB(@LCLCTB(LCLCNS),@TRANS$ARRAY$1,DATA_SIZE);
813  6                         TRANS$1$RDY = FALSE;
814  6                         CALL TRNMIT_PKT(U01DAT);
815  6                         CALL SRVC_TAB_HSKP(5);
816  6                       END;
817  5                     END;

818  4                     DO;     /*  CASE TWO   */
819  5                       IF TRANS$2$RDY = TRUE THEN DO;
821  6                          CALL SNDSEQ(@TP_25, SIZE(TP_25));
822  6                          CALL MOVB(@LCLCTB(LCLCNS),@TRANS$ARRAY$2,DATA_SIZE);
823  6                          TRANS$2$RDY = FALSE;
824  6                          CALL TRNMIT_PKT(U02DAT);
825  6                          CALL SRVC_TAB_HSKP(5);
826  6                        END;
827  5                      END;

828  4                      DO;    /*   CASE THREE    */
829  5                        IF TRANS$3$RDY = TRUE THEN DO;
831  6                           CALL SNDSEQ(@TP_26, SIZE(TP_26));
832  6                           DATA$THREE = TRUE;
833  6                           CALL MOVB(@LCLCTB(LCLCNE),@TRANS$ARRAY$3,DATA_SIZE);
834  6                           TRANS$3$RDY = FALSE;
835  6                           CALL TRNMIT_PKT(U03DAT);
836  6                           CALL SRVC_TAB_HSKP(5);
837  6                         END;
838  5                       END;

839  4                       DO;      /*   CASE  FOUR    */
840  5                         IF TRANS$4$RDY = TRUE THEN DO;
842  6                            CALL SNDSEQ(@TP_27, SIZE(TP_27));
843  6                            DATA$FOUR = TRUE;
844  6                            CALL MOVB(@LCLCTB(LCLCNS),@TRANS$ARRAY$4,DATA_SIZE);
845  6                            TRANS$4$RDY = FALSE;
846  6                            CALL TRNMIT_PKT(U04DAT);
847  6                            CALL SRVC_TAB_HSKP(5);
848  6                          END;
849  5                        END;

850  4                   END;
851  3                   ELSE DO;
852  4                       CALL SNDSEQ(@TP_28, SIZE(TP_28));
853  4                       STATTB(L_RO_DEST_ERR) = STATTB(L_RO_DEST_ERR) + 1;
854  4                       CALL SRVC_TAB_HSKP(5);
855  4                   END;
856  3              END;
857  2      END ROUTE_OUT;
```

```
                   /*****************************************************************************/
                   /*    THIS IS THE MAIN BODY OF THE PROGRAM                                     */
                   /*****************************************************************************/

 1          BAUD$LSB = 00001000B;
 1          BAUD$LSB = 00000000B;
 1          BYTES$SENT$1 = 0;
 1          BYTES$SENT$2 = 0;
 1          BYTES$SENT$3 = 0;
 1          BYTES$SENT$4 = 0;
 1          DATA$THREE = FALSE;
 1          DATA$FOUR =  FALSE;
 1          TRANS$1$RDY = TRUE;
 1          TRANS$2$RDY = TRUE;
 1          TRANS$3$RDY = TRUE;
 1          TRANS$4$RDY = TRUE;

 1        ENABLE;
 1          CALL SNDSEQ(@STARTUP_HDR, SIZE(STARTUP_HDR));
 1          CALL SNDSEQ(@TP_1, SIZE(TP_1));
 1          CALL INIT_L_TAB;
 1          CALL SNDSEQ(@TP_2, SIZE(TP_2));
 1          CALL INIT_U_SHTAB;
 1          CALL SNDSEQ(@TP_3, SIZE(TP_3));
 1          CALL INVINT;
 1          CALL SNDSEQ(@TP_4, SIZE(TP_4));
 1          FOREVER = TRUE;
 1          DO WHILE FOREVER;
 2            CALL SNDSEQ(@TP_5, SIZE(TP_5));
 2            CALL ROUTE_IN;
 2            CALL SNDSEQ(@TP_22, SIZE(TP_22));
 2            CALL ROUTE_OUT;
 2            CALL SNDSEQ(@TP_35, SIZE(TP_35));
 2          END;
 1          CALL SNDSEQ(@TP_36, SIZE(TP_36));
 1        END MAIN;

MODULE INFORMATION:

     CODE AREA SIZE     = 10C1H   4289D
     CONSTANT AREA SIZE = 0924H   2340D
     VARIABLE AREA SIZE = 25BEH   9662D
     MAXIMUM STACK SIZE = 0020H     32D
     1805 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
```

VITA

William F. Matheson was born on August 18, 1944 in Worcester, Massachusetts. He graduated from David Prouty High School, Spencer, Massachusetts in 1962. In September 1962 he entered the U.S. Air Force and was trained as an Electronic Cryptographic Maintenance Technician. In 1977, while assigned to Offutt Air Force Base, Nebraska he received the Bachelor of General Studies degree from the University of Nebraska at Omaha. In 1977 he was selected for the Airman Education and Commissioning Program. In December 1978 he was awarded his Bachelor of Science in Electrical Engineering from the University of Nebraska, Lincoln, Nebraska. He was commisioned in April 1979 through the Officer Training School and was assigned as a Systems Project Engineer at HQ Electronic Security Command, Kelly Air Force Base, Texas. He entered the Air Force Institute of Technology in June 1982.

Permanent Address: 29 Grant Street
Spencer, Massachusetts 01562

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

AD-A138 118

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GE/EE/83D-42	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Air Force Institute of Technology 6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code) Wright Patterson AFB, Ohio 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification) see block 19	

12. PERSONAL AUTHOR(S)

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
MS Thesis	FROM ______ TO ______	1983 December 15	160

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB. GR.
9	2	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Intel 8086 Microprocessor; Network Interface; Intel 8089 I/O Processor; Protocols; Local Area Networks; Local Computer Networks

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: CONTINUED DEVELOPMENT OF A UNIVERSAL NETWORK INTERFACE DEVICE USING THE INTEL 8086 AND 8089 16-BIT MICROPROCESSORS

Advisor: Dr. Gary B. Lamont

Approved for public release: IAW AFR 190-17.
Lynn E. Wolaver 7 Feb 84
... Professional Development
Air Force Institute of Technology (ATC)
Wright-Patterson AFB OH 45433

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Gary B. Lamont	22b. TELEPHONE NUMBER (Include Area Code) (512) 2553450 22c. OFFICE SYMBOL AFIT/ENG

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

Abstract: This research describes the development of a Universal Network Interface Device (UNID II) which is intended to function as a node in a computer communications network. The UNID II is a 16-bit, 8086 microprocessor based version of the present 8-bit Z80A UNID being developed at the Air Force Institute of Technology (AFIT). The UNID II's architecture was based on a conceptual block diagram design presented in a previous AFIT thesis. It is comprised of two modules: a local module, which interfaces the UNID II to host computers and/or peripheral devices; and a network module, which interfaces the UNID II to a computer communications network. In this report the detailed design of the network module, and the construction and testing of the local module is documented. The network module was designed using a pair of 8089 Input/Output Processors in a remote configuration. The local module consists of an Intel SBC 86/12A single board computer and a wire wrap card with four low speed I/O ports. Testing was done using an Intel ICE-86A/88A In-Circuit Emulator. The tests conducted, verified the proper operation of the local module, including some software to process X.25 formatted frames. The UNID II was not tested in a computer communications network environment.